VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2009 **Patrik Dubec**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Webový server pro Magic the Gathering
Web server for Magic the Gathering

2009 **Patrik Dubec**

VŠB - Technická univerzita Ostrava
Katedra informatiky

Fakulta elektrotechniky a informatiky
Akademický rok 2008/2009

Zadání bakalářské práce

Student: **Patrik Dubec**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Webový server pro Magic the Gathering
Web Server for Magic the Gathering

Zásady pro vypracování:

Hlavním cílem bakalářské práce je implementovat webový server pro sběratelskou karetní hru Magic The Gathering. Cíle práce lze shrnout do těchto bodů.
1. Primární funkcí tohoto serveru bude umožnit jednotlivým hráčům hrát proti sobě.
2. Sekundárně by měl pak server poskytovat i další funkce, které souvisí se správou hracích karet a sdílením informací. Uživatel bude mít nějakou svou virtuální sbírku karet a virtuální peníze, za které si bude moci karty pořizovat, například prostřednictvím nějaké „aukce".
Z hlediska praktické implementace bude aplikace vytvářena v jazyce C# s využitím technologií ASP.NET a AJAX. Je také vhodné, aby výsledná aplikace byla vícejazyčná.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Marek Běhálek**

Datum zadání: 30.11.2008
Datum odevzdání: 07.05.2009

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr.Ing. Eduard Sojka
vedoucí katedry

prof. Ing. Ivo Vondrák, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 28.4.2009 Patrik Dubec

Poděkování

Děkuji tímto svému vedoucímu bakalářské práce Ing. Marku Běhálkovi za odbornou pomoc a konzultaci při vypracování.

Abstrakt a klíčová slova

Abstrakt

Tato práce se zabývá návrhem a implementací webového serveru (WS).WS poskytuje potřebnou funkcionalitu pro správu uživatelů rozdělených do různých rolí s rozdílnými přístupovými právy. Hlavní částí celého WS je možnost hraní MTG uživatelů proti sobě. Jako podpůrný prostředek k získávání nových karet je obchod s kartami. Součástí implementace je analýza zadání (datová, dynamická, funkční) a návrh uživatelského rozhraní. Pro samotnou implementaci je využito nových technologií firmy Microsoft, jako je např. LINQ pro komunikaci s MsSQL serverem nebo aplikačního rámce ASP. NET 3.5 a technologie AJAX. Webová aplikace je vyvíjena v jazyce C#, díky kterému máme možnost plně využít výše zmíněného Frameworku.

Klíčová slova

informační systém, webová aplikace, ASP.NET, MsSQL, AJAX, Magic The Gathering

Abstract and key words

Abstract

This project is focused on developing of web application. The application provides necessary set of functions for managing user's accounts divided into groups with different access permissions. The most important feature is possibility of playing a card game against another user. You can use a card shop for getting new cards. There are analyses (data, dynamic, function) and suggestion of user's interface as parts of implementation. There are used new technologies like LINQ for communication with MsSQL server or framework ASP .NET 3.5 and AJAX for development. The application is evolved of using programming language C# that takes advantages of the framework.

Key words

information system, web application, ASP.NET, MsSQL, AJAX, Magic The Gathering

Seznam použitých symbolů a zkratek

HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
WS	Webový server
LINQ	Language Integrated Query
ASP. NET	Active Server Pages
Prototype	je zdokumentovaná otevřená knihovna napsaná v jazyce javascript, která nabízí API pro usnadnění vývoje dynamických webových aplikací.
jQuery	je další zdokumentovaná otevřená knihovna napsaná v jazyce javascript, která nabízí API pro usnadnění vývoje dynamických webových aplikací.
RDBMS	Relational DataBase Management Systém
CSS	Cascading Style Sheets
Use Case	je diagram případů užití. Popisuje dynamickou funkčnost aplikace.
MsSQL	Microsoft SQL Server
DAL	Data Access Layer
BLL	Business Logic Layer
UI	User Interface
Master page	je vzorová stránka sloužící ke sjednocení vzhledu obsahových stránek
Framework	je softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovnu API, návrhové vzory nebo doporučené postupy při vývoji.
Cookie	(anglicky koláček) se v protokolu HTTP označuje malé množství dat, která WWW server pošle prohlížeči, který je uloží na počítači uživatele. Při každé další návštěvě téhož serveru pak prohlížeč tato data posílá zpět serveru.
DOM	Document Object Model
XML	eXtensible Markup Language
AJAX	Asynchronous JavaScript and XML

Obsah

1. Úvod

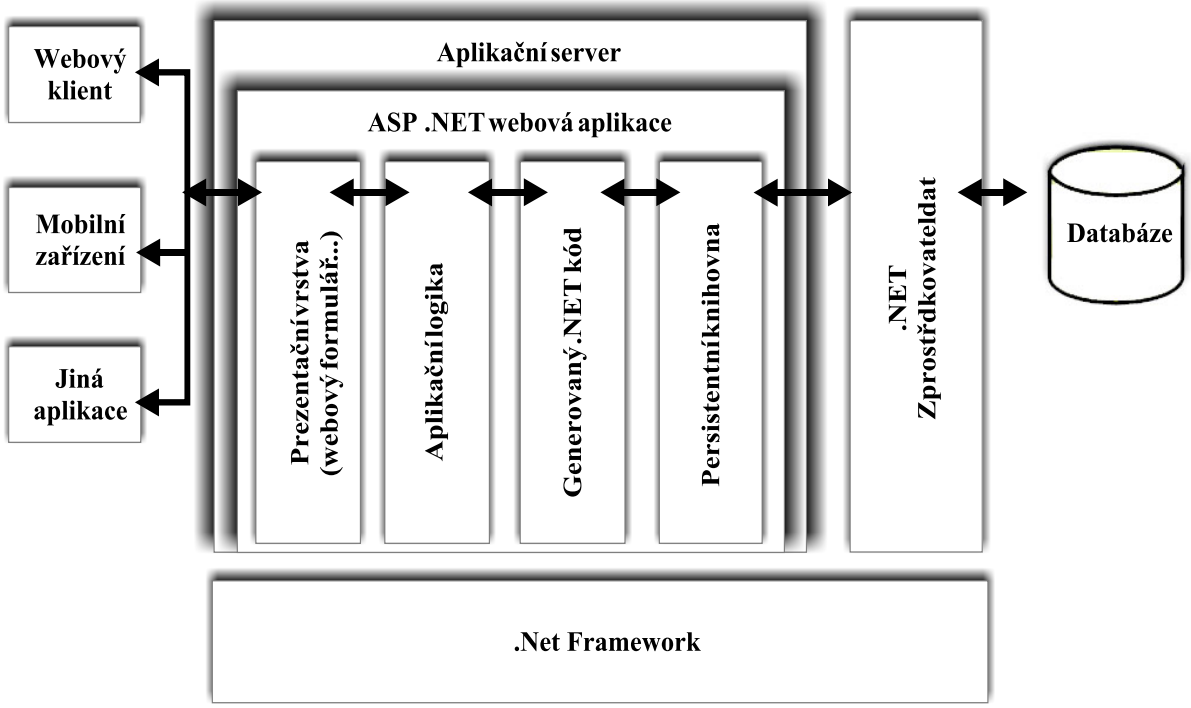
Cílem této práce je vytvořit webový server umožňující hrát karetní hru MTG z kteréhokoli počítače připojeného k internetu. Klub stolních her potřebuje zlepšit přípravu svých členů na turnaje a umožnit jim zdokonalovat se v herní taktice i mimo omezený vyhrazený čas v klubovně. Dále chce přilákat nové hráče do svých řad a nejlepší formou reklamy pro dnešní mládež je interaktivní aplikace volně dostupná na internetu.

Práce je rozdělena do několika kapitol. 2. kapitola ve zkratce popisuje použité informační technologie. V dalších dvou kapitolách se věnuji zadání a rozboru systému na nejvyšší úrovni. V kapitole 5 a 6 se zaobírám analýzou systému a návrhem implementace. V poslední kapitole naleznete popis implementace.

2. Použité informační technologie

2.1. Aplikační rámec ASP .Net 3.5

Technologie Microsoft ASP.NET není jen další generací prostředí ASP (Active Server Pages). Poskytuje zcela nový model programování dynamických webových aplikací. Nabízí širokou škálu tzv. ovládacích prvků, díky kterým máte pocit, jako byste programovali běžnou aplikaci pro stolní počítač. Kód webové aplikace je kompilovaný, což přináší vyšší rychlost pro koncové uživatele. Obrovskou výhodou této platformy je způsob, jakým se generuje výsledný vzhled na straně klienta. Aplikační server na základě HTTP požadavku rozhodne, s jakým prohlížečem komunikuje a odpověď přizpůsobí jeho parametrům.



Obrázek 1: Aplikační rámec ASP .NET

2.2. LINQ (Language INtegrated Query)

Jde o velice zajímavou technologii určenou pro přístup k datům. Je postavena na velmi abstraktních základech, proto má široký záběr možného použití. Pro popis této technologie jsem vycházel z prvních částí dokumentace, viz (3).

2.2.1. Co je to LINQ

Od svého počátku bylo možné v aplikacích postavených nad platformou .NET pracovat z relačními či hierarchickými daty a to hned několika způsoby. V případě dat, která jsou uložena v relační databázi, je možné přistupovat skrze známé rozhraní *ADO .NET* a to jak v připojeném tak odpojeném režimu. Pokud jsou data reprezentována hierarchicky v populárním formátu XML lze využívat typů z assembly *System.Xml* a pracovat s těmito daty buď pomocí DOM, nebo je zpracovávat sekvenčně. Nyní ovšem přichází technologie, která umožňuje zcela nový způsob práce s daty na platformě .NET.

2.2.2. Co LINQ přináší

LINQ přináší podporu dotazování přímo do .NET programovacího jazyku. LINQ lze využít v jazycích C# 3.0 nebo Visual Basic 9, které přináší sadu nových klíčových slov pro provádění LINQ dotazů nad daty. Další velmi příjemnou novinkou, která plyne z integrace dotazování přímo do .NET programovacího jazyku, je odhalení chyb již v době kompilace, takže vaše dotazy budou kontrolovány hned a ne až při běhu aplikace.

2.2.3. Jak LINQ funguje

Abychom LINQ mohli používat ve výše zmíněných jazycích patřících do rodiny .NET, musela být tato platforma obohacena o novou funkcionalitu.

- **Klíčové slovo var -** Nezbytné pro využití anonymních tříd. Nejde o var jako např. v jazyce Javascript, kde je možné var proměnnou použít jednou jako číslo, a poté klidně jako řetězec. Překladač jazyka C# přesně ví, o jaký datový typ se jedná, i když je pro programátora třeba skrytý, takže jde stále o silně typovou konvenci.
- **Lambda funkce –** jde o zjednodušení zápisu anonymních metod (např. var List = poleCisel.Select(n => n > 3); vrátí čísla z pole, která jsou větší než 3)
- **Anonymní třídy -** umožňující rychlé vytvoření objektů přenášejících informace vyžádané z databáze přes LINQ. (např. var person = new {Name = „Pepa“, Age = 10};)
- **Výrazové stromy –** umožňuje objektovou reprezentaci napsaných výrazů, nebo funkcí. Bez této vlastnosti by LINQ nefungoval.
- **LINQ to SQL –** Zajišťuje objektově-relační mapování pro platformu .NET. Překládá LINQ dotazy na T-SQL. Nabízí nám poměrně vysoký výkon (ztráty oproti čistému ADO .NET jen v řádu několika procent). Dotazy jsou implementovány nad typem IQueryable<T>. Pro přístup k datům slouží třídy odvozené od DataContext (náš datový zdroj). Je možné používat uložené procedury formou typových metod, které vám nabídne Visual Studio 2008 jako ostatní běžné metody. Je implementována podpora dědičnosti entit.
- **Formát dotazu –**

IEnumerable<T> q = ***from*** *[typ] proměnná* ***in*** *datový_zdroj*
[where] *podmínka_restrikce*
[orderby] *klíč_řazení [**a/de scending**]*
select *co_se_má_vrátit;*

3. Zadání

Systém by měl nabízet potřebnou funkcionalitu pro hru dvou hráčů proti sobě. Aby tak mohli učinit, je zapotřebí nějakým způsobem získat karty do svého vlastnictví. Z vlastních karet si pak každý hráč vybuduje svůj herní balíček, se kterým bude moci hrát. Uživateli musí být nabídnuta možnost zbavit se svých již dále nepotřebných karet a získat zpět část „peněz“, za které kartu pořídil. Z důvodu vedení statistik by bylo vhodné uchovávat, kromě základních údajů o uživateli, také údaje osobního charakteru, jako jsou bydliště, pohlaví a datum narození. Tyto údaje však nebudou striktně vyžadovány. Za účelem spravování uživatelských účtů by měla existovat pověřená osoba s vyššími pravomocemi, která by tak mohla činit. Tato osoba by dále měla oprávnění přidávat hrací karty do systému a při té příležitosti nastavit pořizovací cenu karty.

Samotná hra by se měla řídit standardními pravidly MTG, viz příloha [II.]

3.1.1. Současný stav

Klub má již internetové stránky, které jsou však pouze informativního charakteru. Jsou napsány ve statickém HTML bez použití kaskádových stylů a obsahují řadu chyb. Klub má však zaplacen webový hosting, který nabízí podporu aplikačního rámce ASP .NET a je k dispozici i databázový server SQL Server 2008. Klub dále vlastní 2 PC s operačním systémem Windows XP a s trvalým připojením k internetu.

3.1.2. Proč nové řešení

Současné webové stránky nejsou vhodné ani jako základ nové aplikace, a to hned z několika důvodů. Obsahují chyby, díky absenci CSS není možné přijatelným způsobem obměnit současný vzhled. Nová aplikace bude muset generovat obsah dynamicky na straně serveru, což s čistým HTML není proveditelné.

4. Specifikace požadavků

Cílem specifikace požadavků je popsat, co má softwarový systém dělat prostřednictvím specifikace jeho funkcionality. Požadavky dělíme z hlediska funkcionality na dvě základní skupiny (funkční a nefunkční).

4.1. Funkční požadavky

Funkčními požadavky rozumíme požadavky na věcný, problémový obsah systému. Získáme odpovědi na základní otázky týkající se funkcionality systému, jako např. jaké funkce bude systém nabízet, kdo je bude využívat a v neposlední řadě, s jakými daty se bude pracovat.

4.1.1. Kdo bude se systémem pracovat

Guest – uživatel, který není do systému přihlášen. Bude mít možnost prohlížet si databázi karet a v případě zájmu se zaregistrovat a stát se tak plnohodnotným uživatelem.

User – běžný uživatel, který je přihlášen do systému. Bude mít vlastní sbírku karet, se kterou bude moci manipulovat. Karty si bude moci kupovat za své elektronické peníze buď z elektronického obchodu, nebo z aukce. S takto nabytými kartami si bude moci sestavit vlastní herní balíček. Již nepotřebné karty bude moci prodávat. Uživatel bude mít přístup ke svým osobním údajům, které může modifikovat. Uživateli je nabídnuta možnost hrát hru proti jinému přihlášenému uživateli.

Administrator – oproti běžnému uživateli bude mít rozsáhlejší pravomoci. Bude mít přístup k profilům běžných uživatelů, které bude moci zablokovat v případě, že se uživatel chová neadekvátně, popřípadě odblokovat, pokud se uživateli uzamkne účet z důvodu opětovného zadání chybných přihlašovacích údajů. Dále bude spravovat hrací karty. Bude moci vytvořit novou kartu a vložit ji do systému, upravovat již stávající, nebo je mazat.

4.1.2. Vstupy do systému

Vstupy znamenají informace, které mají být v IS evidované. Je to seznam entit a jejich atributů. Jsou základem pro datovou analýzu.

Uživatelé - budeme evidovat nepovinné osobní a povinné přihlašovací údaje vyplněné při registraci uživatele ve webovém formuláři. (uživatelské jméno, heslo, email, jméno, příjmení, datum narození, pohlaví, adresa, jiný kontakt, uživatelská skupina)

Karty - budou vkládány pouze administrátorem ve formuláři. Karty si mohou uživatelé prohlížet, pořizovat si je, prodávat je a hrát s nimi. (název, obrázek, edice, barva, typ, vyvolávací cena, síla a pořizovací cena karty)

Vlastnosti karet – přiřazení vlastností k vybraným kartám

Herní balíčky – uživatel si do své sbírky pořídí karty. Z těchto karet, které patří jen jemu, si bude moci postavit hrací balíček.

Hra – uživatel může založit novou hru. Aby tak mohl učinit, potřebuje mít postavený herní balíček, se kterým bude hrát proti jinému uživateli, který se ke hře připojí.

4.1.3. Výstupy ze systému

Výstupy znamenají výstupní sestavy, které budou uživatelé potřebovat.

Karty obchodu – seznam karet, které je možné si pořídit v elektronickém obchodě s možností detailnějšího pohledu na každou kartu ze seznamu.

Detail karty – detailní pohled na karty, aby bylo možné dobře přečíst její text.

Karty uživatele – seznam karet, které má uživatel ve svém vlastnictví s možností detailnějšího pohledu na každou kartu ze seznamu.

Hrací balík – seznam uživatelových hracích balíčků s možností prohlédnout si obsažené karty.

Úprava hracího balíku – pohled na vybraný balíček s možností úprav jeho obsahu

Uživatelé – seznam uživatelů dostupný jen administrátorům s možností editace uživatelů.

Detail uživatele – Zobrazení všech informací vybraného uživatele s možností editace jeho atributů.

Hra – obrazovka, kde budou karty z balíků dvou hráčů.

Seznam her – seznam dostupných her, ke kterým se může uživatel připojit.

Seznam vlastních her – seznam her, které uživatel sám založil, a čekají na soupeře.

4.1.4. Tabulka funkcí

akce uživatele	aktér	reakce systému
Přidání karty	Administrator	Do tabulky karet se zapíše nový záznam.
Editace karty	Administrator	V tabulce karet se edituje příslušný záznam.
Prodat kartu	User	Uživatel má možnost prodat kartu za část její pořizovací ceny.
Koupit kartu z obchodu	User	Karta se přidá do uživatelovy sbírky karet.
Připojit se ke hře	User	Uživatel se připojí k založené hře a hra začne.
Hraní hry	User	Uživatelé hrají hru proti sobě.
Založit hru	User	Hra se přidá do seznamu založených her, kde se k ní mohou uživatelé připojit.
Vytvořit hrací balík	User	Do tabulky balíků se přidá nový baliček s id uživatele.
Editovat hrací balík	User	Přidá/odebere karty do/z uživatelova balíčku.
Registrace	Guest	Do tabulky uživatelů se přidá nový záznam.
Editace uživatele	Administrator	Aktualizují se informace o uživateli.
Editace osobního profilu	User	Aktualizují se informace o uživateli.
Smazání uživatele	Administrator	Odebere se záznam z tabulky uživatelů.
Zablokování uživatele	Administrator	Změní se status uživatele, který se nebude moci přihlásit.

Tabulka 1: Tabulka funkcí

4.2. Nefunkční požadavky

Mimo věcnou náplň požadavků na systém existují i požadavky týkající se okolností a způsobu řešení. Mezi takovéto požadavky můžeme zahrnout nároky na použití programovacích jazyků, případně dodržení různých standardů.

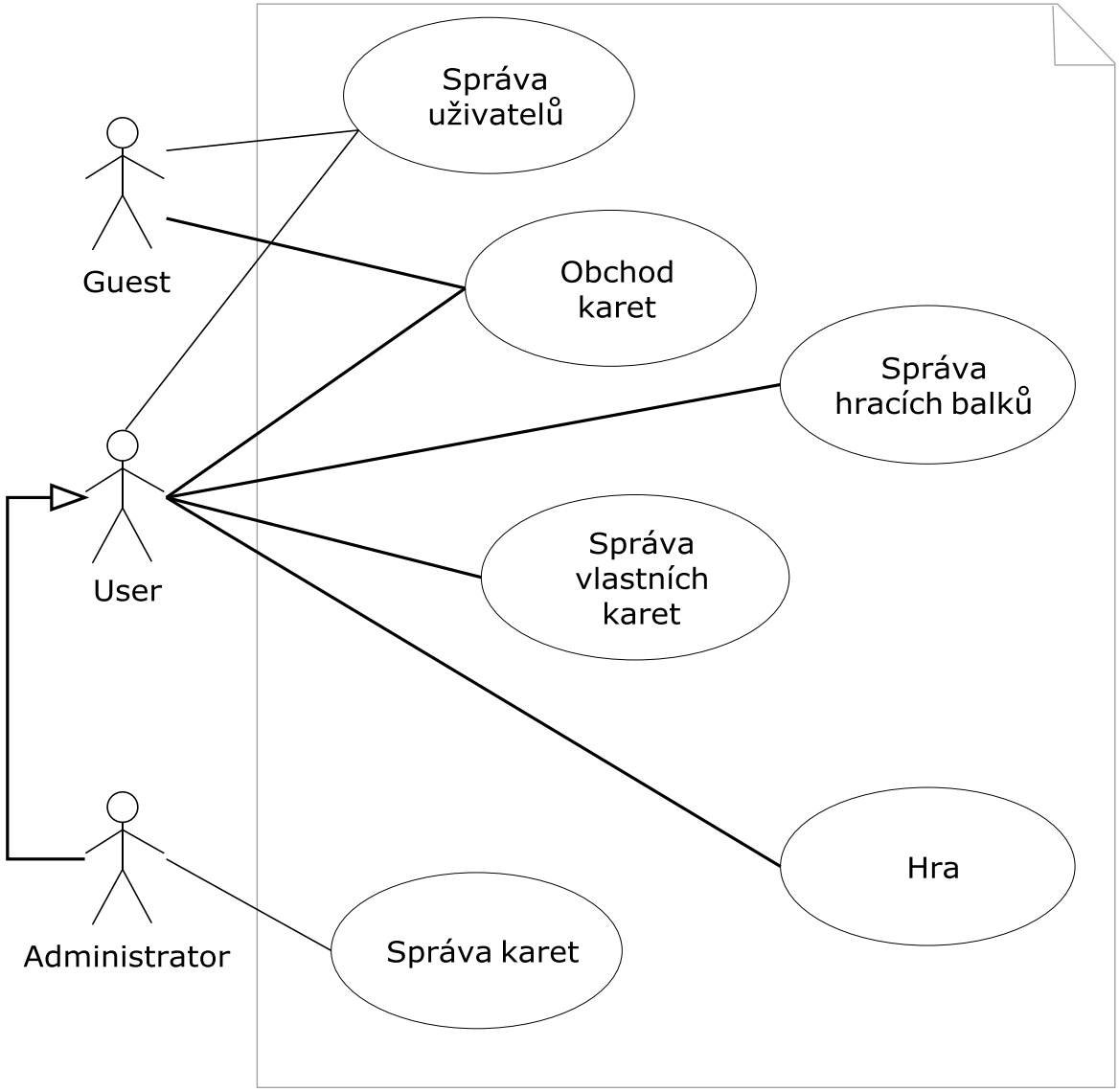
1. Využít stávající webový hosting tzn. MsSQL 2008
2. Optimalizovat pro internetový prohlížeč Mozilla Firefox 3.0 a Internet Explorer 7.
3. Uživatelské rozhraní by mělo být jak česky, tak anglicky.
4. V současné době má klub 16 členů. Zvolit takové technologie, aby byl zajištěn bezproblémový chod aplikace pro alespoň dvojnásobný počet uživatelů.

4.3. Diagramy případů užití (Use case)

Use Case model je grafickým zobrazením dynamické struktury systému. Popisuje vztahy mezi funkcemi systému a jeho aktéry, jaké hlavní úlohy nebo funkce mají být prováděny konkrétním uživatelem nebo skupinou uživatelů.

4.3.1. Obecné funkce systému

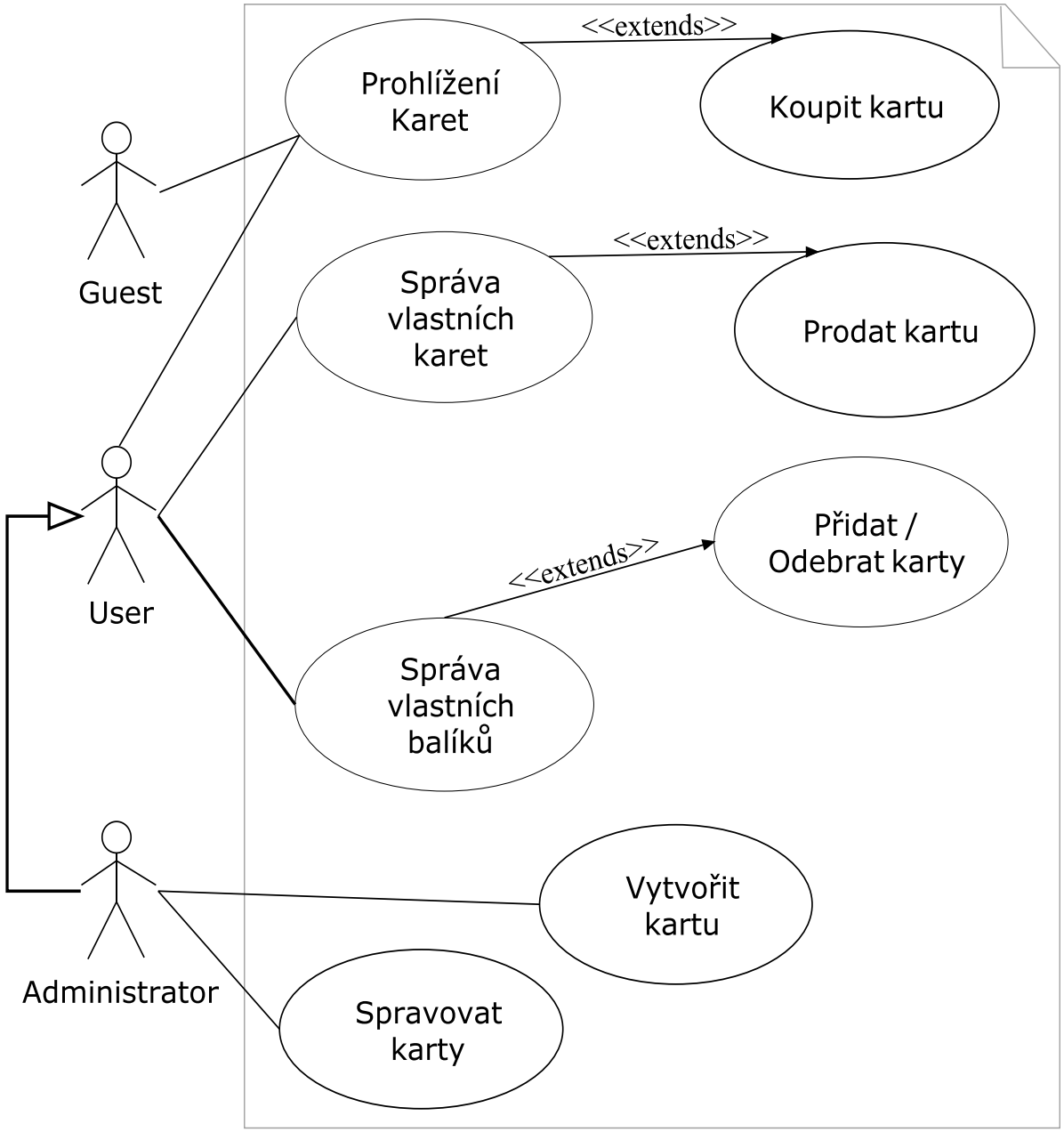
V první fázi je vhodné rozdělit celý systém na několik modulů na základě jejich rozličné množiny funkcí, kterou budou uživatelům nabízet. Budeme přitom vycházet ze zadání, konkrétně z funkčních požadavků systému. Dále určíme, které uživatelské skupiny budou mít k danému modulu přístup a které ne.



Obrázek 2: Diagram případů užití - obecné funkce

4.3.2. Karty a uživatelé

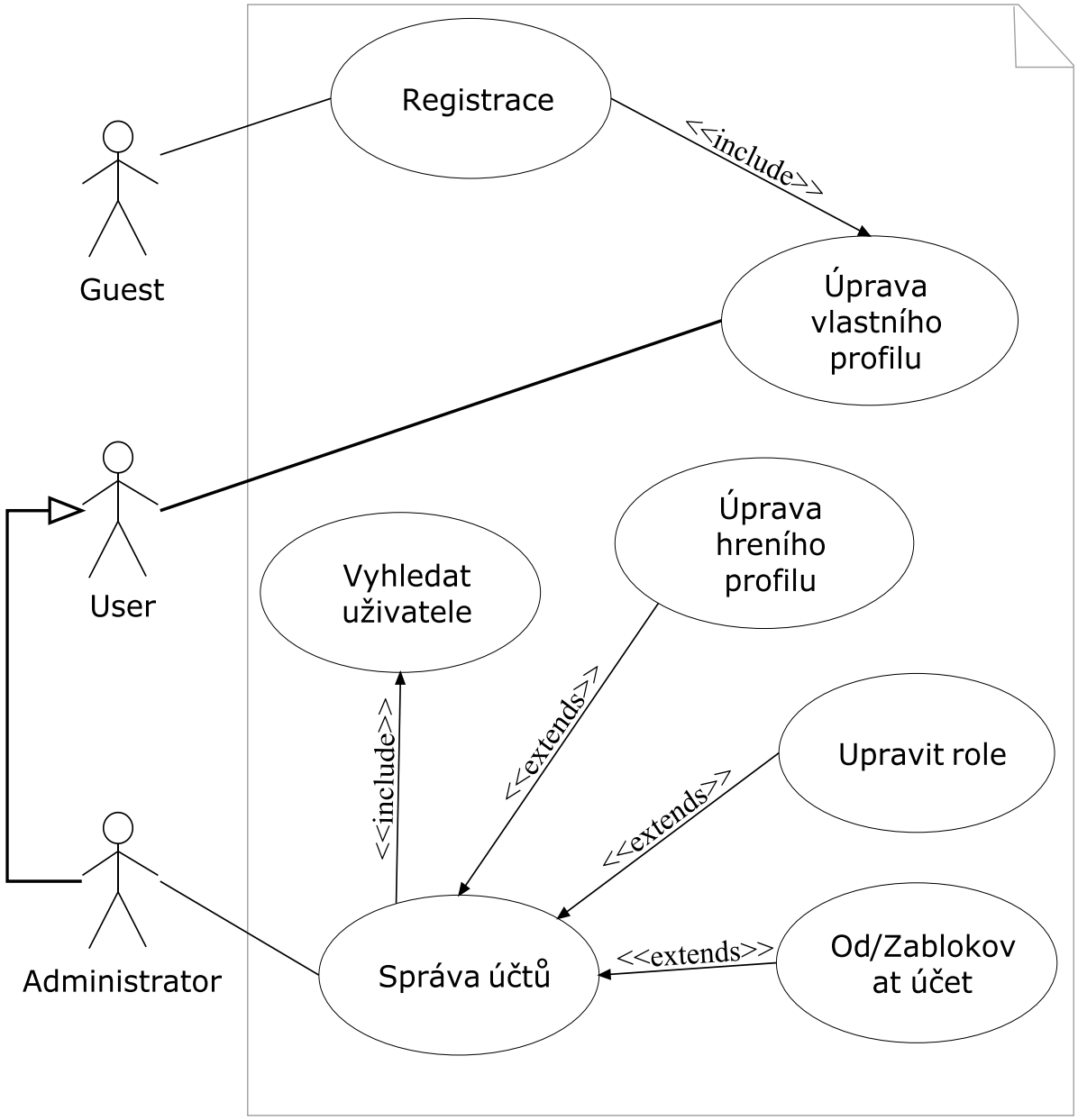
Jednou z nejdůležitějších částí celé aplikace je vazba karet na uživatele. Karty je nezbytné do systému nejprve vložit. Databáze karet je pro prohlížení k dispozici všem uživatelům. Aby mohl uživatel s kartami hrát, musí si je koupit, a tak je získat do osobního vlastnictví. Pokud má uživatel dostatek karet, může si vybudovat hrací balíček. Jako se i v reálném životě můžeme snadno zbavit nepotřebných věcí, tak by měl mít i náš uživatel možnost, v rámci aplikace, prodat své karty.



Obrázek 3: Diagram případů užití - karty a uživatelé

4.3.3. Správa uživatelů

Nedílnou součástí aplikace je samozřejmě správa uživatelů. Návštěvníci webu se mohou zaregistrovat. Tím získají jednoznačnou identitu v rámci aplikace a navíc se jim zpřístupní další funkce systému. Uživatel s oprávněními administrátora by měl být navíc schopen např. zablokovat účet nepohodlným uživatelům, nebo naopak odblokovat účet těm, kteří zadali opakovaně špatné heslo při přihlašování.



Obrázek 4: Diagram případů užití - správa uživatelů

5. Analýza systému

V informatice je předmětem analýzy aplikační oblast reálného světa. V tomto případě bude potřeba důkladně prostudovat pravidla hry. Na základě analýzy se většinou provádí implementace nového systému.

5.1. Datová analýza

Datová analýza je základem pro takové systémy, kde databáze, ukládání dat a vyhledávání informací z ní jsou hlavním účelem, tedy pro informační systémy.

5.1.1. Lineární zápis

Díky poměrně velikému počtu typů entit jsem pro definici atributů nezvolil třídní diagram, který by po obohacení o atributy pozbyl své primární výhody – přehlednosti, ale sáhl jsem po řešení pomocí lineárního zápisu. Nebudu zde uvádět kompletní výčet typů entit a jejich atributů, ale pro názornost jen pár zástupců. Zbytek je uveden, formou datového slovníku spolu s typy a integritními omezeními atributů, v příloze [I.].

Lineární zápis posloužil k popisu vlastností jednotlivých typů entit. Jinak řečeno ukazuje, jaká data je třeba o objektech uchovávat, aby neztratily žádný ze svých rysů. Atributy jsou zvoleny tak, aby veškeré tabulky splňovaly podmínky 3. normální formy. To především znamená, že jsou odstraněny redundance, což má za následek zachování integrity shromažďovaných dat.

Typy entit

[**primární klíč**, cizí klíč]

Uživatelé (**id**, id_aplikace,uživatelské_jméno, anonymní, poslední_aktivita)

Uživatelé_Mají_Karty (**id**, id_uživatele, id_karty, množství)

Karty (**id**, název, expanze, vyvolávací_cena, vzácnost, typ, útok, obrana, obrázek, pořizovací_cena)

Balíčky (**id**, id_uživatele, název)

Hry (**id**, čas_založení, čas ukončení, kolo_prvního_hráče, fáze_kola)

Karty_Ve_Hře (**id**, id_karty, id_hráče, pozice_ve_hře, útok, obrana)

Hráči (**id**, stav_životů, červená_mana, zelená_mana, modrá_mana, bílá_mana, černá_mana, bezbarvá_mana)

Uživatelé_Hrají (**id**, id_uživatele, id_hry, id_hracího_balíčku, id_hráče, zakladatel, zvítězil, remíza)

Vyvolávací_Ceny (**id**, červená_mana, zelená_mana, modrá_mana, bílá_mana, černá_mana, bezbarvá_mana, tapni, obětuj, životů, odhoď)

Efekty (**id**, typ_efektu, přidej_útok, přidej_obranu, lízni_karet, odhoď_karet, znič, uděl_zranění, přidej_životů, přeruš_kouzlení, přidej_červené_many, přidej_zelené_many, přidej_modré_many, přidej_černé_many, přidej_bílé_many, přidej_bezbarvé_many)

5.1.2. Konceptuální schéma

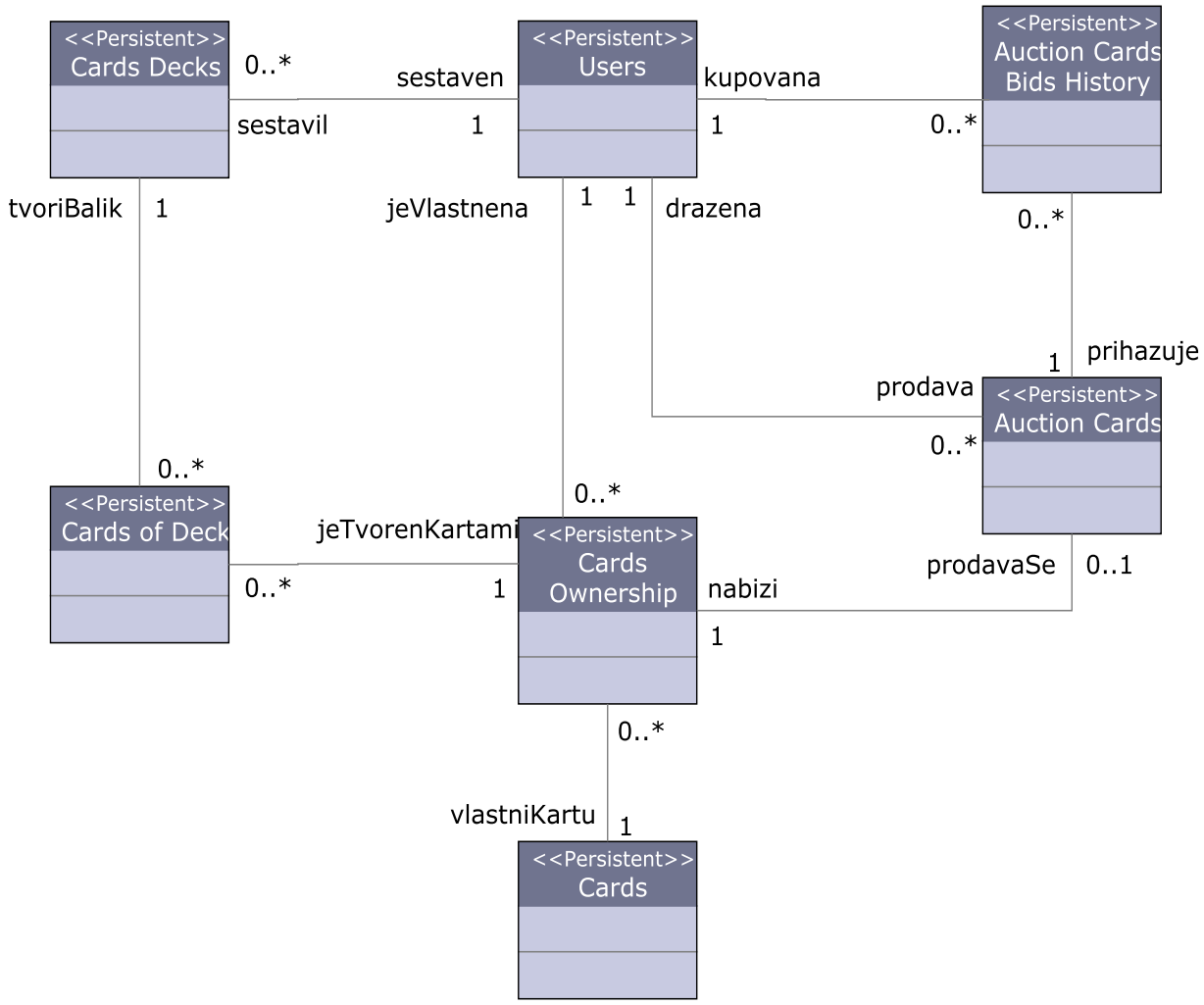
Pro grafické znázornění vztahů mezi entitami (přesněji mezi typy entit) jsem se rozhodl využít prostředků jazyka UML, namísto klasického mapování pomocí E-R diagramu. Podle literatury (1) jsou ekvivalentní formou zápisu třídní diagramy. Pro větší přehlednost je vynechán seznam atributů jednotlivých typů entit, které je opět možno dohledat v přiloženém datovém slovníku [I.].

Nyní, když již mám základní představu o tom, jakou funkcionalitou bude aplikace oplývat, je nejvyšší čas promyslet, jaká data k tomu budou zapotřebí. Definuji si proto typy entit, které budou reprezentovat množiny objektů stejného typu a vztahy mezi nimi. To umožní vytvoření popisu dat v databázi nezávisle na jejich uložení. Tento konceptuální model představuje formální popis modelované reality.

Následující příklady popisují nejdůležitější objekty celého systému a vztahy mezi nimi. Neuvádím zde například diagramy tabulek pro samotné uživatelské účty a profily, které jsou sice podstatné, ale věřím, že většina programátorů se již s problémem uchovávání uživatelských dat setkala.

Uživatelé a karty

Stojím před důležitým problémem, a to sice v jakém vztahu bude přihlášený uživatel s kartami. Systém musí vědět, které karty a v jakém počtu uživatel vlastní. Z těchto vlastních karet si poté může postavit balíček. Jako v reálném světě, tak i zde je možné jednu a tu samou kartu použít ve více různých balíčcích.



Obrázek 5: Třídní diagram - uživatelé a karty

Karty

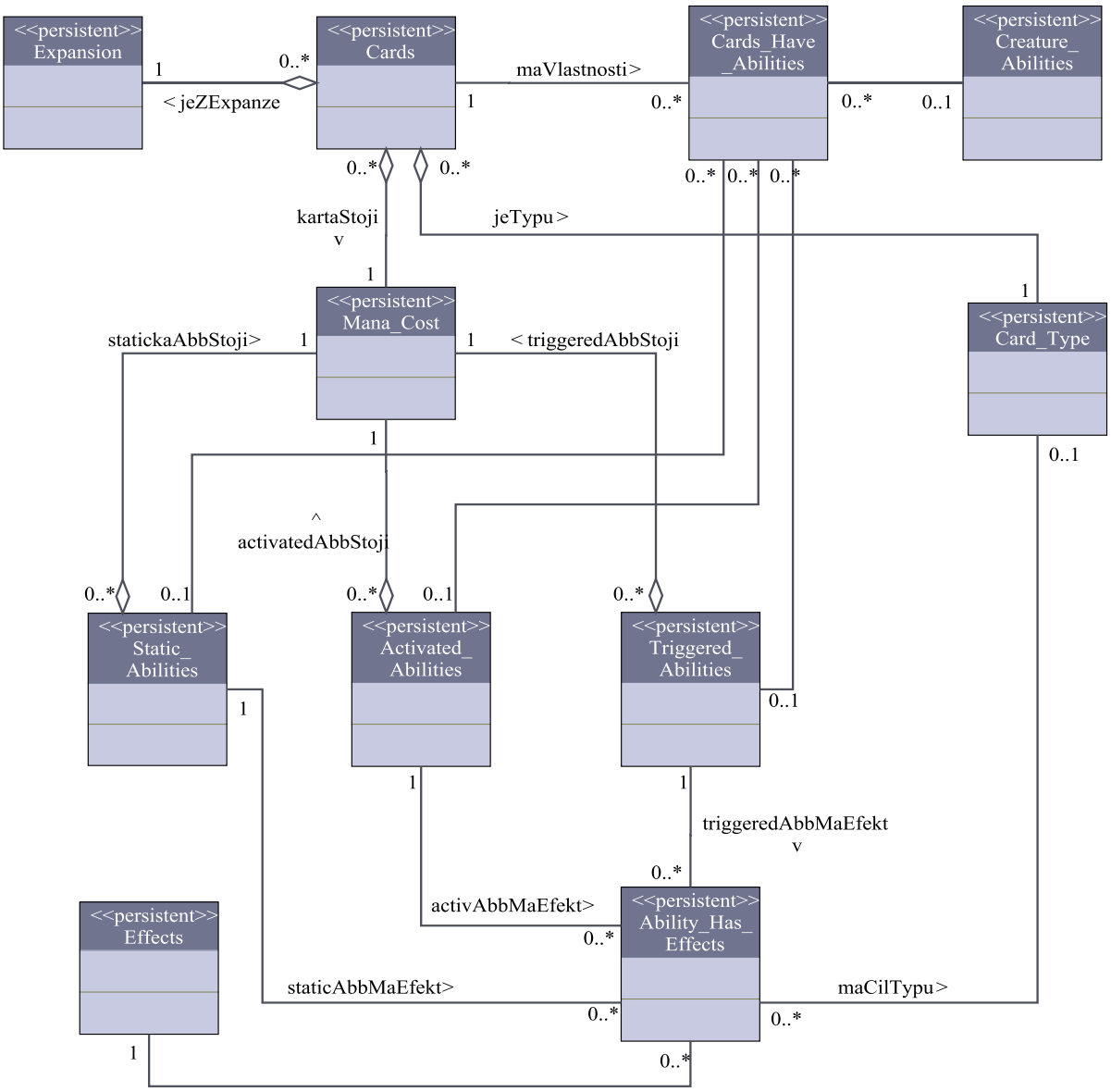
Asi nejsložitějším úkolem bylo analyzovat složení samotných karet na hraní. Ve hře Magic The Gathering existuje spousta různých karet a díky tomu i hromada nejrůznějších vlastností. Pro rozbor problému bylo třeba zcela důkladně porozumět filozofii samotné hry a dopodrobna znát pravidla. Proto bylo nezbytné nastudovat oficiální příručku ke hře, kterou naleznete v příloze [II.]. Na základě nabytých vědomostí jsem vysledoval podobné rysy, a ty jsem rozdělil do několika skupin.

Nejprve se karty dělí na základě svého typu na země, příšery, kouzla a očarování. Každá skupina karet jednoho z těchto typů disponuje rozličnou škálou vlastností.

Bohužel díky značné rozmanitosti karet se některé vlastnosti vyskytují napříč několika typy. Například země po použití přidá něco jako peníze, za které si můžete vyvolat nějakou příšeru nebo zahrát kouzlo. Tu samou vlastnost můžeme však najít i u některých příšer nebo kouzel, kdy po zaplacení vyvolávací ceny můžeme dostat tyto peníze, ale třeba jiného typu případně množství. Z tohoto důvodu bylo nutné i vlastnosti separovat do skupin. Konkrétně se jedná o skupinu pro

vlastnosti příšer, statických, aktivovatelných a konečně spouštěných vlastností. Vlastnosti se liší nejen tím, jakému typu karty náleží, ale také tím, co jejich použití způsobí a jakým způsobem se vyhodnocují. Tedy i efekty, které vlastnosti působí, je třeba rozdělit.

Na efekty se váže další omezení týkající se cíle, na který je možno jej použít. Za cíl může být považována buď některý konkrétní typ karet (příšery, země aj.), nebo některý z hráčů. A v poslední řadě je ještě důležité mít na paměti, že nic není zadarmo, a tak i za používání vlastností je třeba platit. Za cenu je považováno buď tzv. tapnutí karty (nejde o nic jiného, než že se karta otočí o 90° a až do dalšího kola je dále nepoužitelná), nebo již zmíněné peníze produkované zeměmi různých barev.

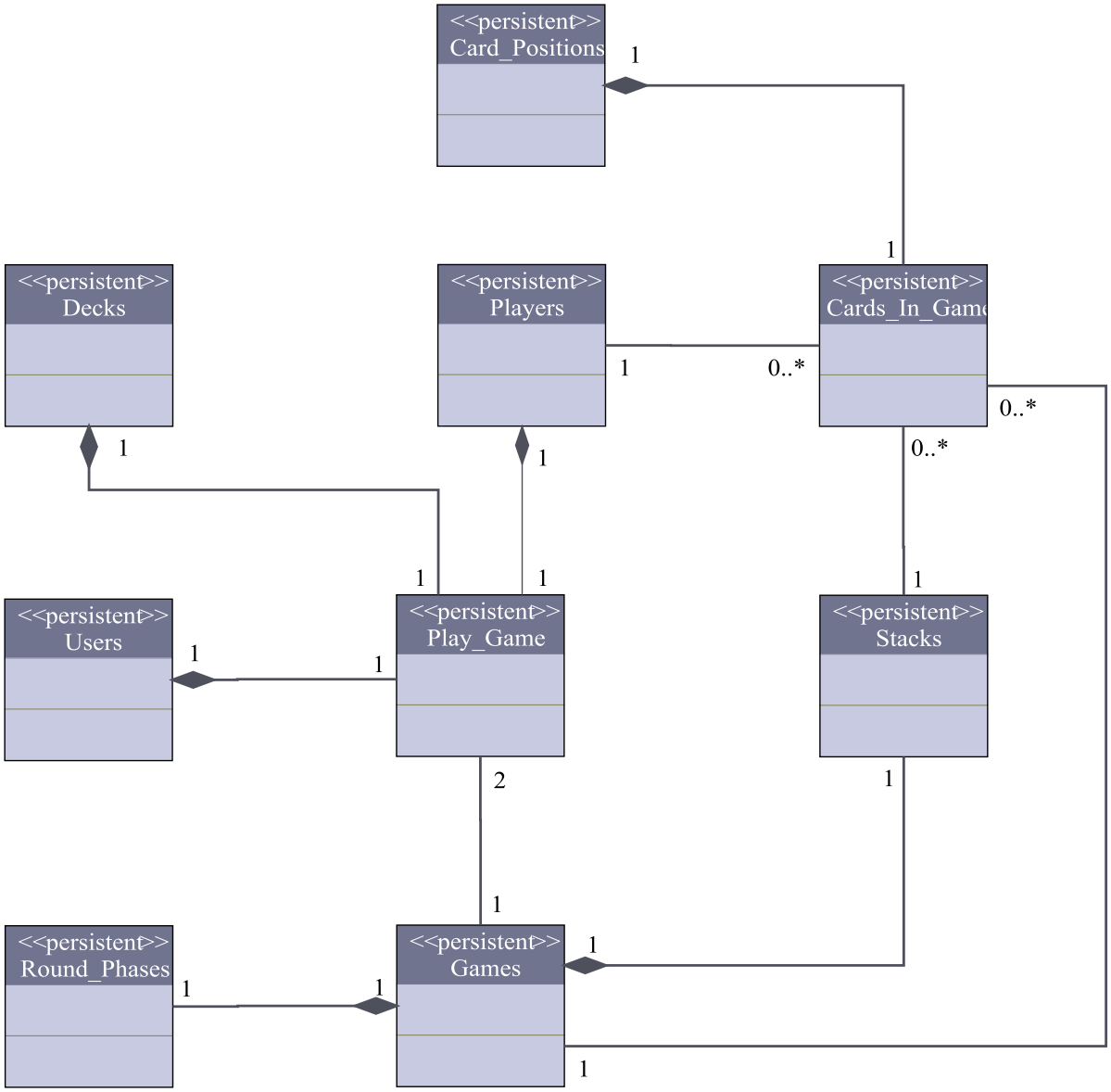


Obrázek 6: Třídní diagram - karty

Hra

K analýze hry bude opět nezbytné vycházet z pravidel. Nejprve se jen ve zkratce zmíním, jak vypadá její průběh. Na začátku si každý hráč zvolí balíček, se kterým bude hrát. Karty se zamíchají a může se začít. Oba soupeři si vezmou z vrchu balíčku 7 karet do ruky. Kolo začíná tím, že si hráč, který je na řadě, vezme další kartu. Poté může karty z ruky vykládat do hry. Cílem hry je ubrat protihráči všech 20 životů, což může učinit pomocí kouzel, nebo útoky svých přišer. Použitá kouzla a mrtvé přišery se odkládají na hřbitov.

Zde vzniká nutnost definovat nové typy entit, jako je například hráč, který má své karty a životy. Každé kolo se musí rozdělit do jednotlivých fází. Herní pole je také tvořeno několika zónami, jako karty v ruce, v balíčku, ve hře, nebo na hřbitově.



Obrázek 7: Třídní diagram - hra

5.2. Dynamická analýza systému

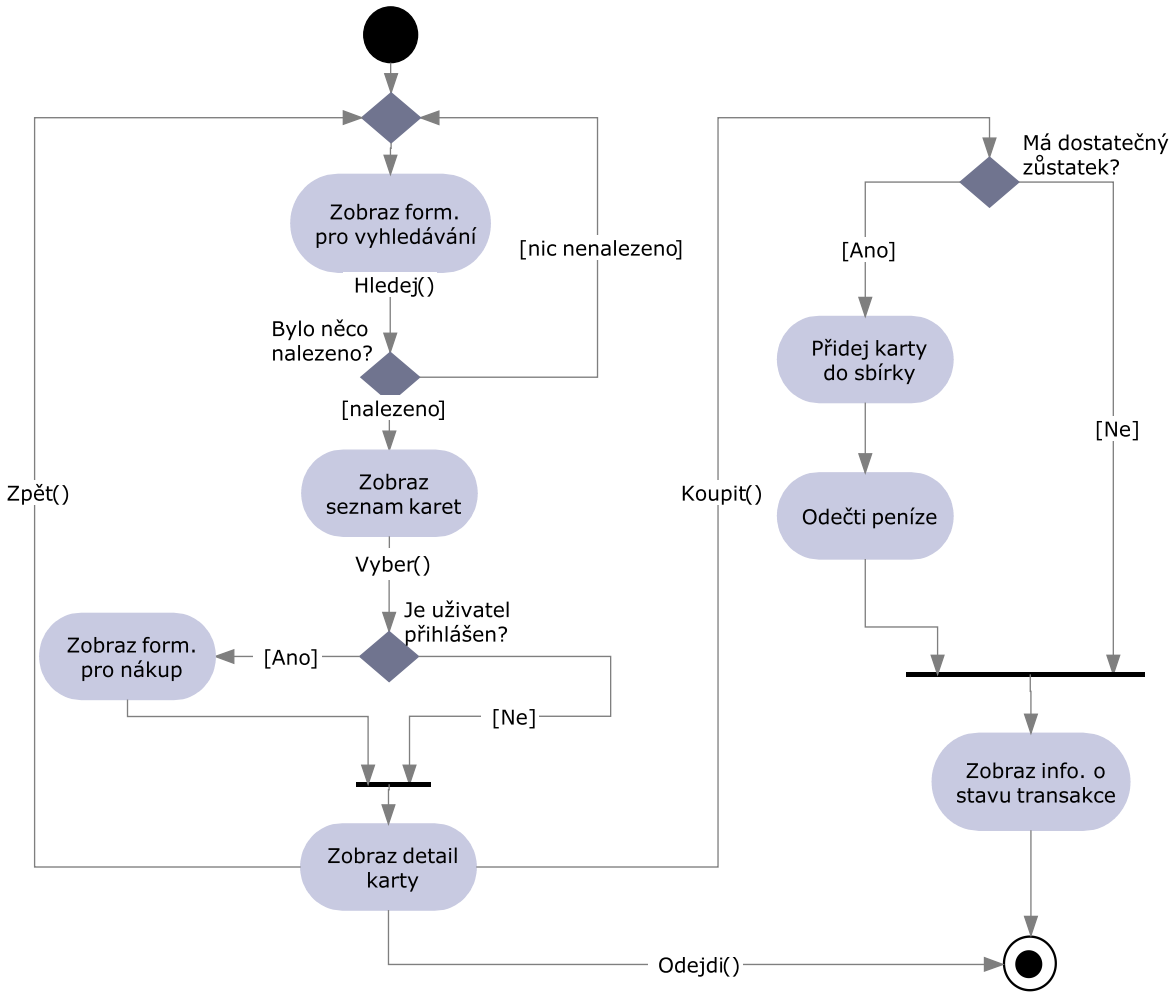
Za úkol má popsat všechny operace, které je zapotřebí s daty v navržené databázi provádět – všechny funkce systému. Obecně to je ukládání, modifikace a rušení dat, výpočty, třídění, vyhledávání informací a formátování výstupních informací.

5.2.1. Aktivitní diagramy

Následující diagramy znázorňují tok řízení z aktivity do aktivity. Ukazují sekvenci stavů, které nastávají v čase, a ukazují podmínky způsobující přechody mezi stavy. Na rozdíl od stavových diagramů, diagram aktivit se soustřeďuje spíše na proces výpočtu než na objekty samotné. Přitom vycházím z diagramů případů užití. Pro ukázku jsem vybral procesy pro vytvoření a koupi karty.

Koupě karty

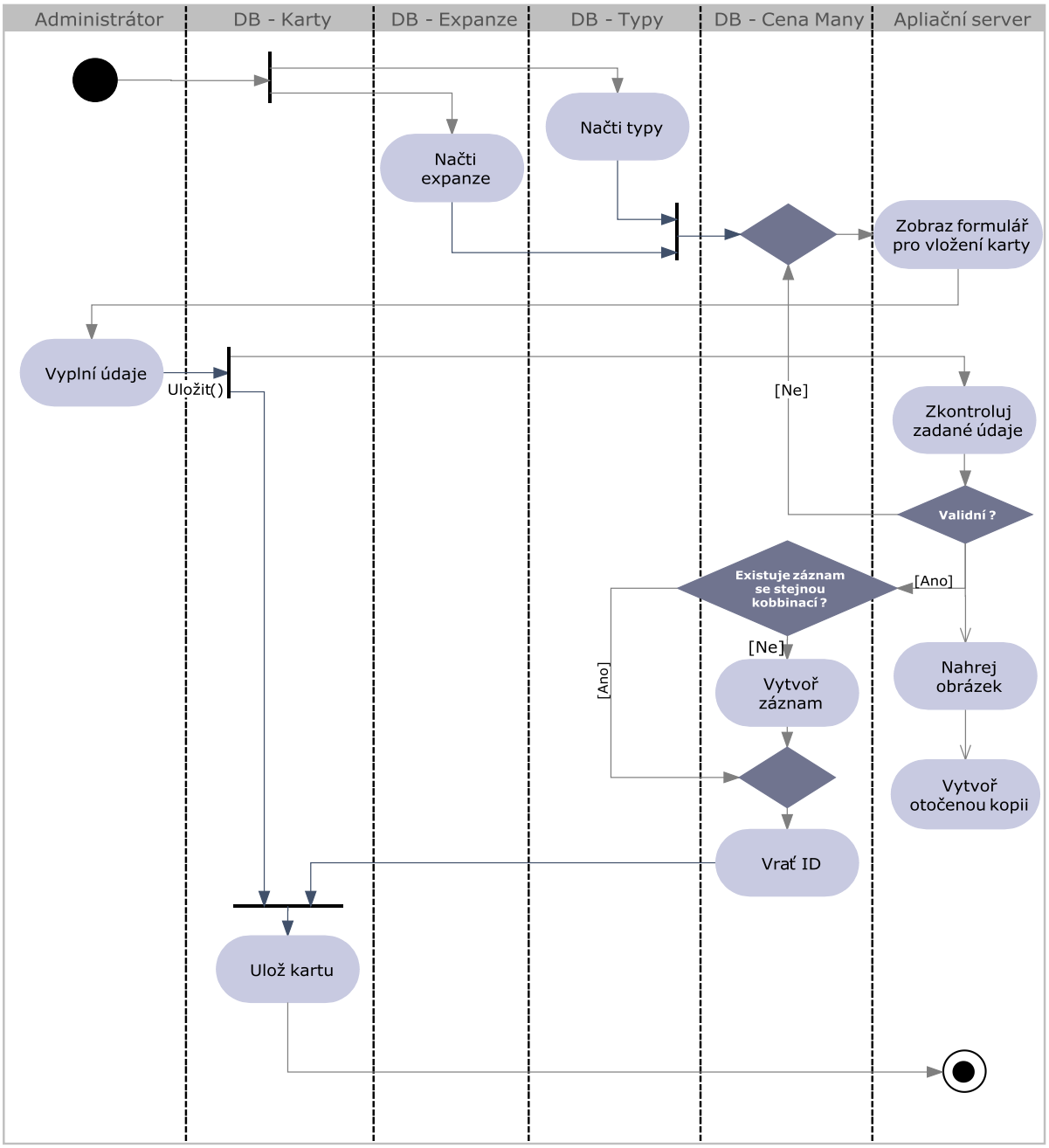
Následující diagram popisuje způsob, jakým si uživatelé mohou pořídit karty. Uživatel musí být nejprve přihlášen a mít dostatečný zůstatek, aby si karty mohl pořídit.



Obrázek 8: Aktivitní diagram - koupě karty

Vytvoření karty

Pro přidání karty do systému musí být dodržen sled několika událostí a je nutno splnit několik podmínek. Nejprve musíme uživateli nabídnout formulář, který obsahuje několik cizích atributů. Před vložením karty do databáze je třeba ověřit správnost zadaných dat a nakonec nahrát obrázek karty na server.



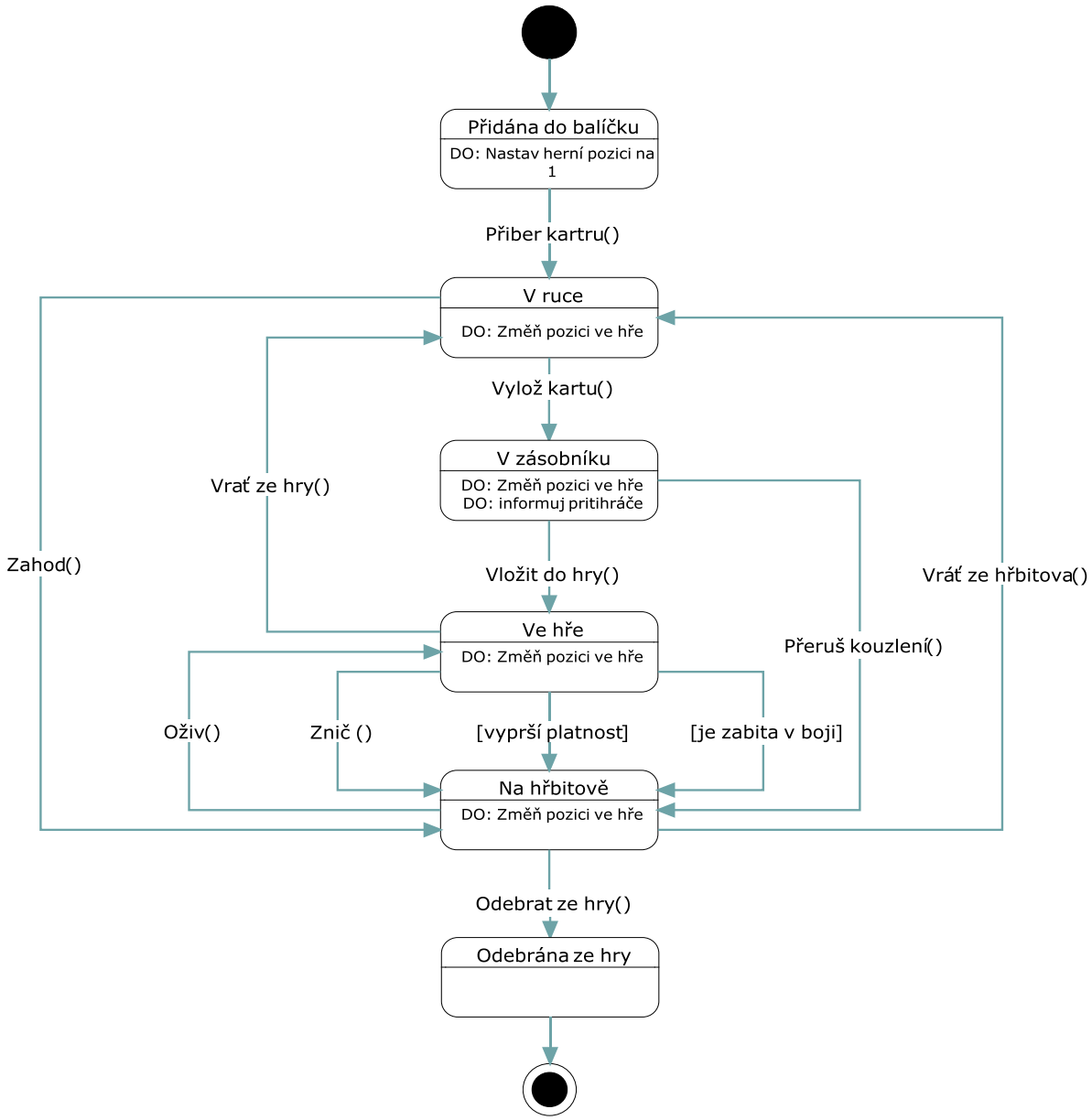
Obrázek 9: Aktivitní diagram - vytvoření karty

5.2.2. Stavové diagramy (STD)

Stavový diagram zobrazuje životní cyklus objektu, události způsobující přechody z jednoho stavu do jiného a akce, které vyplývají z této změny stavu. Na vybraných diagramech níže demonstruji chování dvou pro hru klíčových objektů.

Karta ve hře

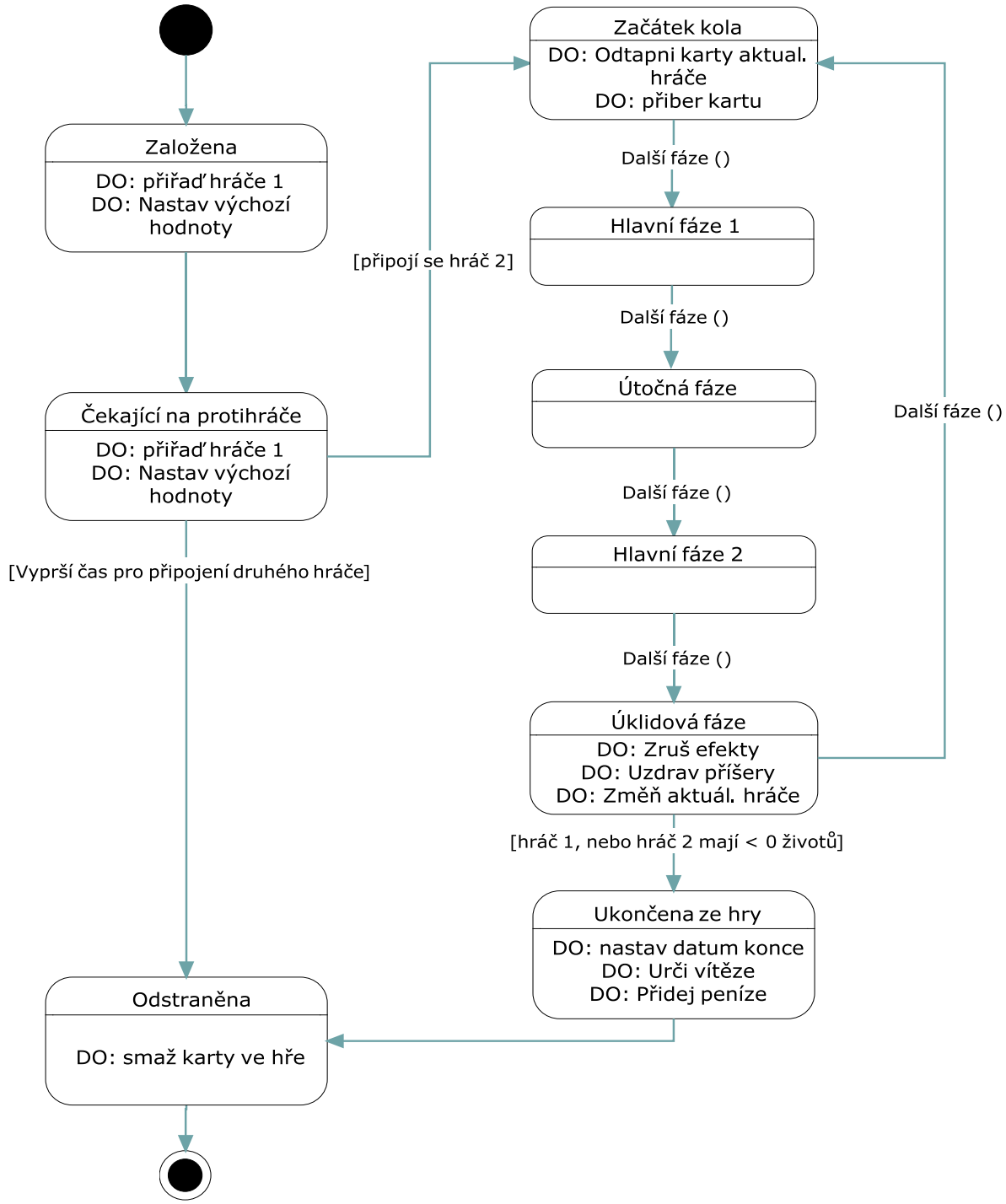
Karta se ve hře může nacházet na několika pozicích, mezi kterými může různě přecházet. Životní cyklus karty ve hře začíná tím, že se zamíchá do balíčku, ze kterého si hráč karty dále bere. Poté je může po zaplacení vyvolávací ceny vložit do hry, kde splní svůj úkol. Když je karta zničena, jde na hřbitov.



Obrázek 10: STD – Karta ve hře

Fáze hry

Životní cyklus hry začíná tím, že se některý z uživatelů rozhodne založit novou hru. Poté čeká, až jiný hráč jeho výzvu příjme. Když se tak stane, hra může začít. Oba hráči se střídají po jednom kole. Každé kolo má navíc několik fází, které na sebe striktně navazují. Hra může být ukončena tak, že jeden z hráčů je zabit, tj. má na konci kola nula nebo méně životů.



Obrázek 11: STD – fáze hry

6. Návrh systému

Výsledkem analýzy je několik modelů budoucího systému. Hlavním úkolem návrhu je projít všechny modely z analýzy a doplnit je řadou implementačních detailů. Dále se provede návrh komunikace s uživatelem. Pro zvýšení přehlednosti jsem do této kapitoly zařadil i prvky, které se při vývoji nachází až ve fázi samotné implementace a při návrhu ještě nejsou k dispozici, jako např. hotové prvky uživatelského rozhraní, nebo ukázky zdrojových kódů.

6.1. Návrh implementace

V etapě návrhu implementace dojde především k upřesnění algoritmů definovaných ve fázi analýzy v závislosti na použitém programovacím jazyce a začlenění aplikačního rámce a ostatních komponent do vlastních tříd. Dalším úkolem je rozdělení systému do modulů na základě funkcionality.

6.1.1. Návrh architektury

Pro implementaci využiji tzv. víceúrovňový návrh. To znamená oddělení kódu pro přístup k datům od kódu aplikační logiky a prezentačního kódu (uživatelského rozhraní), aby se tak dal web lépe udržovat a škálovat. Celý systém tedy rozdělím do několika vrstev. Při volbě tohoto schématu jsem se inspiroval přečtením kapitoly „Návrh webové aplikace“ viz (2), kde se autor podrobně zaobírá výhodami a nevýhodami tohoto postupu.

Datové úložiště

Dle zadání budu používat MsSQL Server 2008. Výhodou tohoto RDBMS je, mimo jiné, excelentní integrace do vývojového prostředí Visual Studia, nebo kvalitní dokumentace.

Vrstva přístupu k datům (DAL)

Tato vrstva obsahuje kód, který k získání, vkládání, mazání a aktualizaci dat spouští dotazy nad databází. Jedná se tedy o kód, který je nejblíže databázi, o níž musí vědět naprosto všechny podrobnosti (tj. např. schéma tabulek či názvy polí). Oddělením kódu pro práci s databází získáme několik výhod:

- Některé dotazy pro získávání dat se používají na více stránkách. Pokud bych je umístil přímo do samotných stránek a dotaz by bylo nutné změnit, musel bych hledat, kde všude je použit, což by mohlo vést k chybám. Díky tomuto přístupu stačí dotaz změnit jednou a samotná stránka nemusí o ničem vědět.
- Pokud by bylo třeba z nějakého důvodu měnit RDBMS, stačí změnit kód jen v této vrstvě a zbytek aplikace by mohl zůstat beze změn.

Při návrhu objektově-relačního (O-R) mapování jsem využil nové technologie LINQ. Postupoval jsem dle návodů a dokumentace viz (3). Vytvořím si tedy novou třídu

MtgClassesDataContext reprezentující objekty mého O-R mapování a definující metody pro vlastní práci s databází prostřednictvím těchto objektů.

Všechny třídy budou navíc dědit z třídy *DataAccess*, kde je vytvořena jediná statická instance třídy MtgClassesDataContext reprezentující objekty O-R mapování. Tento přístup dovolí komunikovat s databází všem třídám této vrstvy, aniž by bylo nutné vždy navazovat nové spojení.

Vrstva aplikační logiky (BLL)

Datová vrstva se skládá ze tříd, které získávají data z databáze a vracejí je jako kolekce instancí vlastních tříd, které slouží pouze jako silně typový kontejner. Vrstva aplikační logiky tato data zpracovává a předává je vrstvě uživatelského rozhraní. Navíc k nim přidává validační logiku a odvozené vlastnosti. Vrstvě uživatelského rozhraní nabízí statické metody pro vytvoření, úpravu, mazání a čtení dat.

Všechny třídy této vrstvy budou potřebovat znát některé stejné informace. Proto je opět vhodné, aby dědily z jedné rodičovské třídy, která poskytuje informace o přihlášeném uživateli (IP adresu, uživatelské jméno, nebo unikátní označení uživatele - ID).

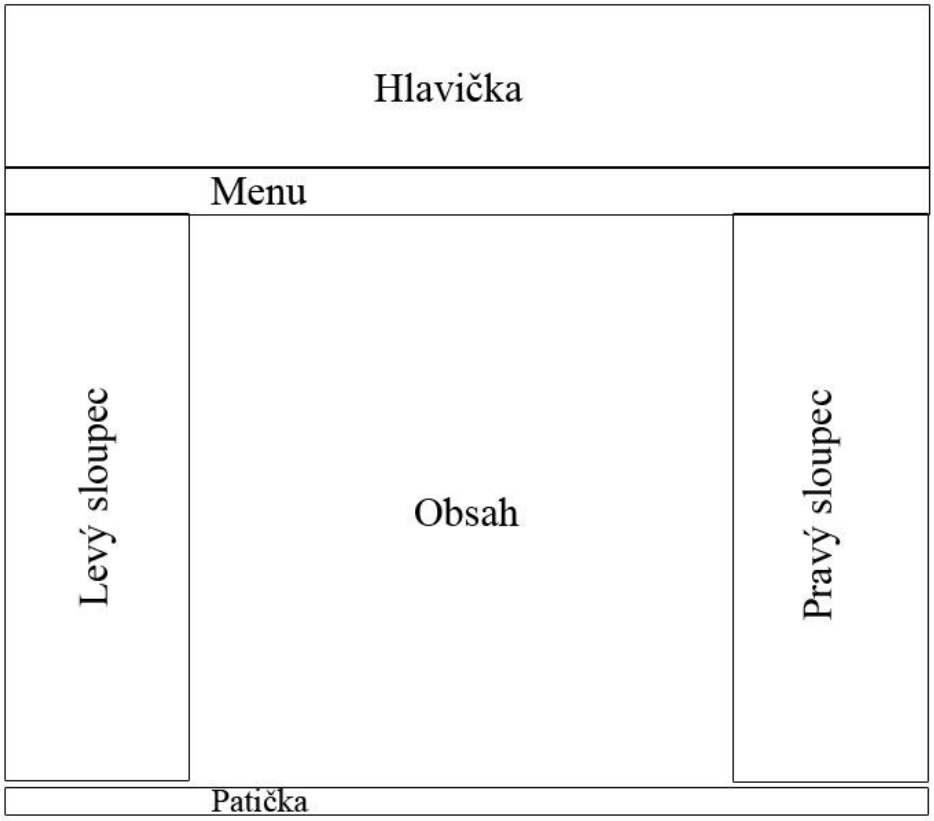
Vrstva uživatelského rozhraní (UI)

Tato vrstva slouží k vizuální reprezentaci dat, která získá od tříd vrstvy aplikační logiky. Kromě vzhledu se kód této vrstvy stará o zachytávání chyb, které by mohly vzniknout za běhu aplikace, a prezentuje je uživatelsky přívětivou formou. K tomu je třeba, aby všechny stránky obsahovaly kód pro zpracování chyb. Nejlepším způsobem, jak toho docílit je opět vytvořit bázovou třídu. Lepší ovšem je snažit se chybám předcházet, proto se tato vrstva bude dále starat o validaci vstupů od uživatele, a to jak na straně klienta pře samotným odesláním požadavku, tak následně na straně serveru.

6.2. Návrh uživatelského rozhraní

Významným rysem aplikace by mělo být atraktivní uživatelské rozhraní (UI). Vzhled je velmi důležitý, protože jde o první věc, které si uživatel všimne. V některých případech může nepříjemný vzhled odradit návštěvníky ještě dříve, než vůbec přejdou k získávání informací, nebo využití funkcionality. Grafika je však jen jedním z aspektů UI. Uživatelé by měli být schopni aplikaci nejen ovládat, ale práce s ní by měla být intuitivní a jednoduchá.

Vzhled celého webu by měl působit kompaktně napříč všemi stránkami. Z tohoto důvodu je vhodné pevně definovat rozložení základních grafických prvků, tzn. navrhnout layout, který bude všemi stránkami dodržován.



Obrázek 12: Návrh UI - layout

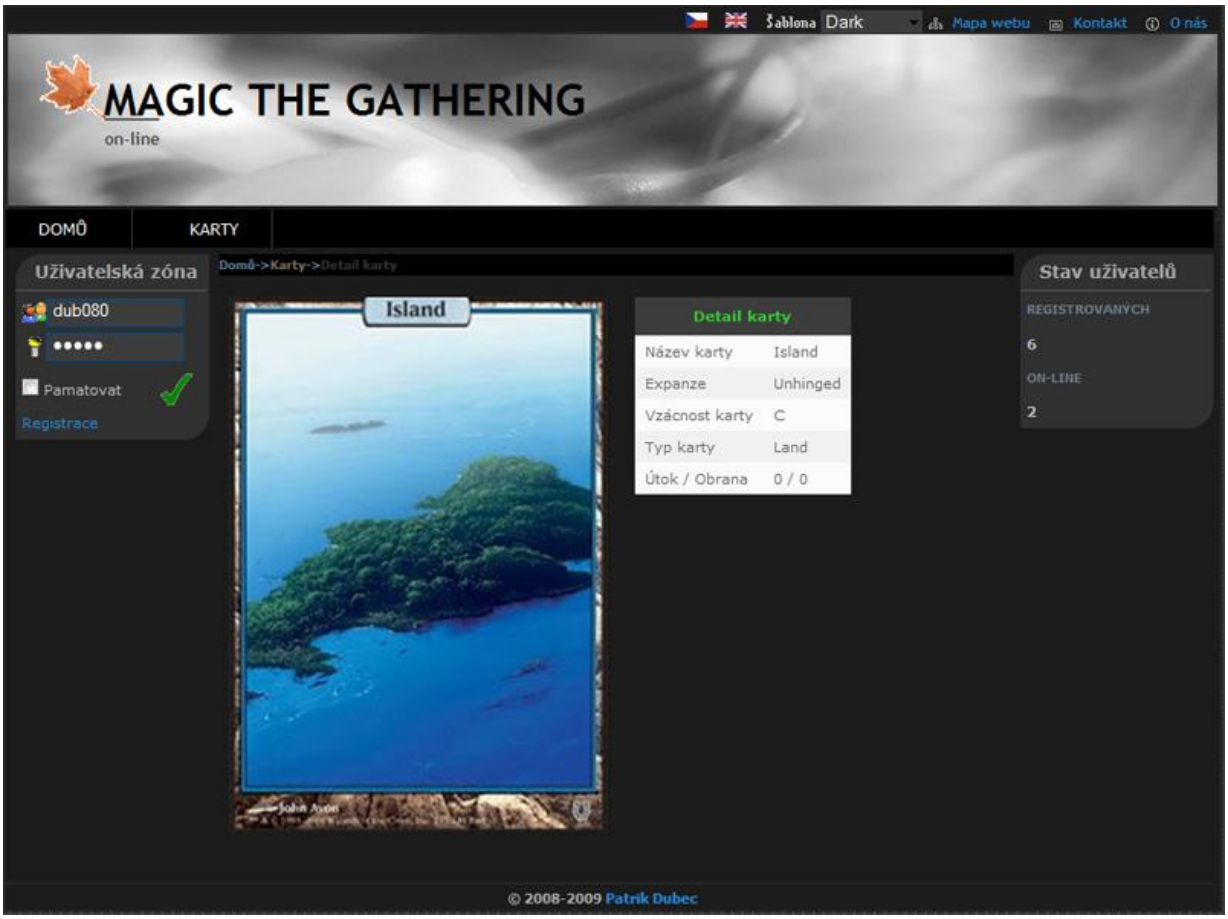
Aby nedocházelo ke zbytečným omylům ze strany uživatelů, je dále důležité sjednotit vzhled formulářů. Tím je myšleno především jednoznačně definovat umístění tlačítek. Pokud je jednou tlačítko pro smazání vlevo a jednou vpravo nejen, že to působí nevzhledně, ale především může snadno docházet k překlepům.

Pro snadnější zpracování nabízených informací uživatelem je dále nezbytné barevně rozlišovat řádky tabulek při dlouhých výpisech. Tyto seznamy by dále měly umožnit rozdělení velkého objemu dat do více stránek, popřípadě umožnit třízení dle atributů.

K zachování jednotného vzhledu všech stránek jsem s výhodou použil prvek tzv. *vzorové stránky (master page)*. Jde o stránku, ve které se definuje základní rozložení grafických prvků (menu, hlavičku, patičku) a vymezí se kontejnery, jejichž obsah se zpřístupní obsahovým stránkám. Při překladu dojde ke spojení obou těchto stránek do jedné.

Každá sekce je vnořena do kontejneru, které formátuji pomocí kaskádových stylů (CSS). Tím docílím toho, že vzhled se dá měnit pouhou editací jediného souboru, se kterým jsou prvky spjaty. Každý uživatel může preferovat jiné vizuálně-stylistické nastavení, a tak si může zvolit jeden z dostupných grafických témat. K tomu je zapotřebí za běhu aplikace vyměnit soubor se styly a při načítání stránek sdělit každé z nich, který soubor použít. Stránka tuto informací získá z tzv. cookie, ve kterém je tento údaj uložen na straně klienta i po té, co uživatel stránku opustí.

Úspěšný moderní web by měl být schopen zpřístupnit svůj obsah široké veřejnosti a neměl by se omezovat pouze na hranice svého domovského státu. Aby tomu tak mohlo vskutku být, je nezbytné umožnit změnu jazykového prostředí. Naštěstí platforma .NET mi práci do jisté míry usnadnila. Změna jazyka funguje obdobným způsobem, jako změna grafických témat. Samotné překlady (jak české, tak i anglické) se nachází ve speciálních XML souborech, ke kterým má web přístup. Při dodržení jmenné konvence těchto souborů můžeme na základě aktuálního jazykového prostředí rozhodnout, který z nich se má použít.



Obrázek 13: Hlavní obrazovka

6.2.1. Herní deska

Při hře mohou mít oba hráči na stole vyloženo i několik desítek karet. Aby tak nevznikl opravdový chaos, je nezbytné vnést do hraní pevný řád. Jako i při normální hře se skutečnými kartami tak i zde by mělo být jasné, které karty patří kterému hráči.

Moderní webové prohlížeče již naštěstí do jisté míry uchovávají obsah načtených stránek ve své dočasné paměti na straně klienta. To má za následek rychlejší nahrávání stránek, které již uživatel v minulosti navštívil. Množství dat, které si takto prohlížeč pamatuje, však jako programátor nejsem schopen ovlivnit. Pro rychlejší běh aplikace tak může být vhodné, omezit opětovné načítání stránek. Při každém výběru karty tedy není vhodné, vracet prohlížeči veškerý obsah znovu. Z tohoto důvodu jsem opět použil technologie *AJAX* (viz (4)), tedy asynchronní zasílání zpráv na server a zpět. Aplikační server je pak schopen, na základě těchto informací, rozhodnout o tom, která karta událost vyvolala a prohlížeči vrátit jen nezbytné množství dat.

Jak již bylo řečeno, karet může být ve hře poměrně hodně. Kvůli tomu je nezbytné do hry umístit jen jejich miniatury. Pro lepší čitelnost se po označení karty zobrazí její detail v levé části obrazovky. Detail karet zůstává pevně ukotven i při posouvání stránky, aby měl hráč informace vždy po ruce. Zmenšením karet se ušetří trocha místa, ale ne dost. Jak tedy mít na stránce ještě více karet a nezmenšovat do nekonečna jejich velikost? Řešení je nasnadě. Karty budou zobrazovány v posuvných kontejnerech. Tím elegantně docílím toho, že karet se do hry vejde neomezené množství, a to při zachování vysoké míry přehlednosti. K softwarovému řešení tohoto problému jsem opět s výhodou použil nástavbu na otevřenou knihovnu *jQuery*, viz (5). Ta posloužila jako základ pro definování těchto kontejnerů. Karty se vyloží do obyčejného listu, který se při načítání stránky zpracuje a převede na DOM, se kterým se pohodlně manipuluje. Poté je třeba zpracovat události po kliknutí na šipku pro posun a do viditelné části dokumentu vložit požadované karty. Nezbytnou součástí je samozřejmě definice stylů pro vzhled kontejnerů.

```
<script type="text/javascript">
//<![CDATA[
        jQuery(document).ready(function() {
            jQuery('#cardsInHandSlider').jcarousel({
                // Configuration
                scroll: 1
                                        ...
            });
        });
//]]>
</script>

<div class="gameMyHand">
<asp:Repeater runat="server" ID="rptCardsInHand"OnItemDataBound="rptBound">
      <HeaderTemplate>
      <ul id="cardsInHandSlider" class="jcarousel-skin-tango">
      </HeaderTemplate>

      <ItemTemplate>
      <li><a href="#" onclick="Update(<%# Eval("CardInGameID") %>)">
      <asp:Image runat="server" ImageUrl='<%# Bind("ImgURL") %>'/>
      </a></li>
      </ItemTemplate>

      <FooterTemplate>
      </ul>
      </FooterTemplate>
```

```
</asp:Repeater>
</div>
```

Obrázek 14: Herní deska

6.2.2. Úprava hracího balíku

K úpravě uživatelova balíků jsem zvolit odlišné řešení, než jen další výpis seznamu karet do tabulky. Lidský mozek se lépe orientuje podle grafických obrazců, než podle textů. Když uživatel prochází karty, které by si chtěl přidat do svého hracího balíčku, snáze si vybaví, jaké má karta vlastnosti podle miniatury. Tím se omezí počet přepínání na detail karty a omezí se tak počet dotazů na databázi.

Aby se karet na stránku vešlo co nejvíce, je nutné omezit prostor, který zabírá každá z nich. Z tohoto důvodu nebylo vhodné ke každé kartě zvlášť přidávat vlastní tlačítko, kterým by se přidávala do balíčku. Situaci jsem vyřešil tak, že každou kartu v seznamu jsem vložil do kontejneru, který jsem, pomocí otevřené knihovny *Prototype* (viz (6)), převedl na objekt, se kterým se dá v prohlížeči libovolně manipulovat.

```
<div class="draggable" id="<%# Eval("ID") %>">
<table width="100px">
                                        ...
      <tr class="rowlight">
            <td colspan="2">
            <mtg:ImagePopup runat="server" ID="MyPopupImage"
                  ImageUrl='<%# Eval("ImgURL") %>'
                  NavigateUrl='<%# Eval("ImgURL") %>'
                  ToolTip='<%# Eval("Name") %>' />
            <td>
       </tr>
```

```
</table>
</div>

<script type="text/javascript">
                new Draggable('<%# Eval("ID") %>', {
                  revert: true,
                  scroll: window
                                        ...
                });
</script>
```

Dále bylo zapotřebí definovat další kontejner, do kterého se budou karty přetahovat. Po přesunutí na požadované místo se serveru pošle asynchronní zpráva s jednoznačnou identifikací karty. Díky tomu se nemusí znovu načíst celá stránka, pouze se patřičně upraví seznam karet. Stejně se tak tomu děje i při úpravě počtu karet. Způsob řešení navíc poměrně zatraktivňuje vzhled aplikace.

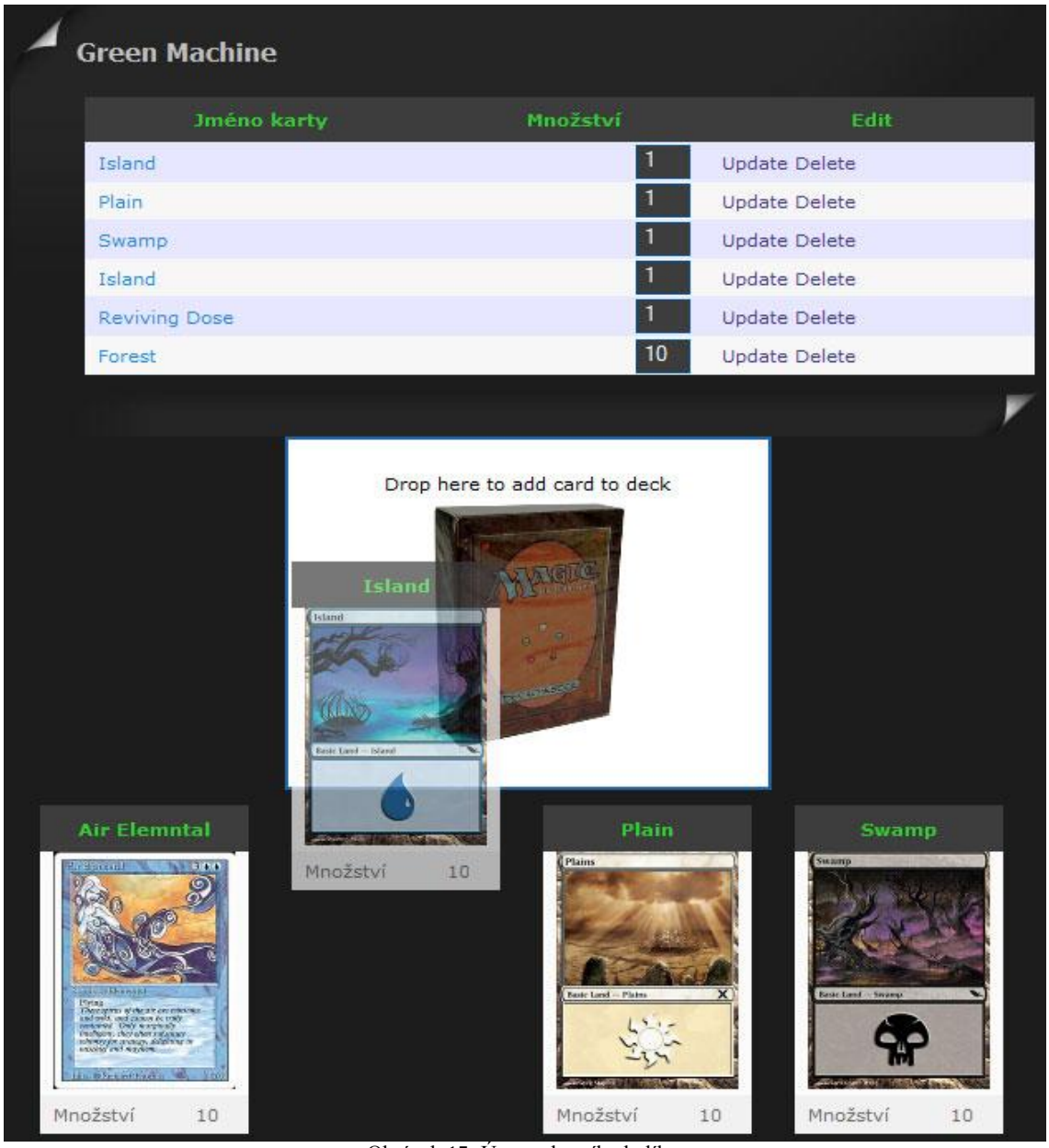
```
<div id="fixeddiv">
      <asp:Label runat="server" ID="Label2" Text="Drop here…" ToolTip=""
      ForeColor="Black"/><br />

      <asp:Image runat="server" ID="Image2"
      ImageUrl="~/Images/cardDeck.jpg" Width="125px" Height="165px"/>
</div>

<script type="text/javascript" language="javascript">
// <![CDATA[
      var prm = Sys.WebForms.PageRequestManager.getInstance();

      //sends dropped card ID to server to update cards of deck
      function Update(draggable) {
            prm._doPostBack('upCards', draggable.id);
      }

      Droppables.add('fixeddiv', {
            accept: 'draggable',
            hoverclass: 'hover',
            greedy: true,
            onDrop: function(draggable, droppable, event) {
                Update(draggable);
                droppable.highlight({
                    startcolor: '#000000',
                    endcolor: '#ffffff'
                });
            }
        });
// ]]>
</script>
```



Obrázek 15: Úprava herního balíku

6.3. Modul pro koupi karty

V této části uvádím příklad návrhu a implementace modulu, který se stará o kupování karet. Algoritmus samotné koupě je již pospán pomocí minispecifikace níže (Tabulka 2: Minispecifikace - koupě karty). Třídní digram znázorněný na obrázku (Obrázek 16: Schéma vztahů mezi třídami jednotlivých vrstev) ukazuje strukturu modulu napříč všemi vrstvami aplikace. Ostatní moduly aplikace jsou vyvíjeny podobným způsobem.

Minispecifikace

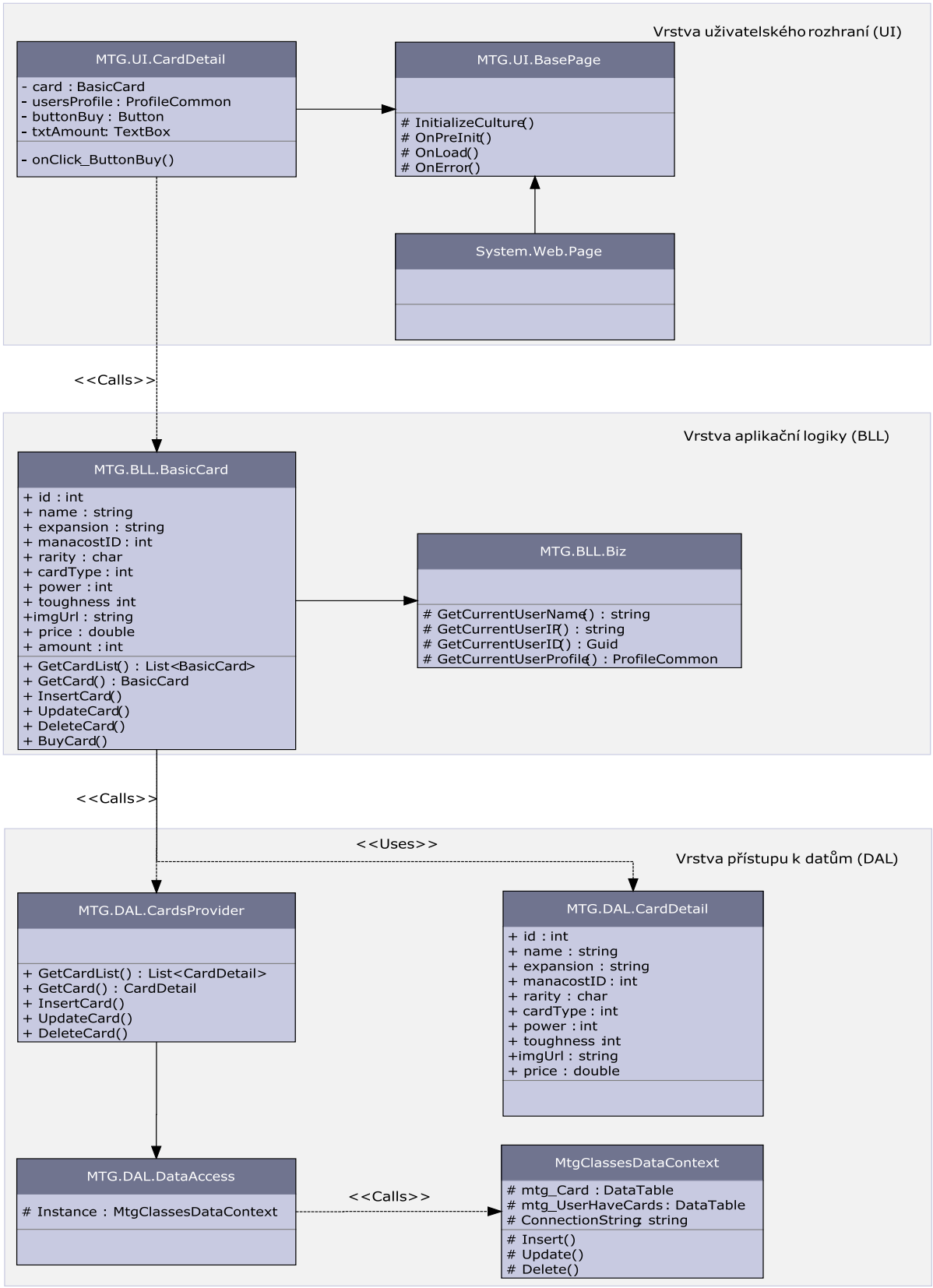
Minispecifikace slouží k detailnímu popisu elementárních funkcí na posledním stupni hierarchického rozkladu. Při tvorbě popisu vycházím z dříve navržených aktivitních diagramů.

Minispecifikace – Koupě karty
Účastníci: User, Aministrator
Vstupní podmínky: Uživatel je přihlášen do systému.
Tok událostí: 1. Zobraz seznam všech dostupných karet z tabulky Cards. 2. Uživatel si vybere požadovanou kartu. 3. Do p.IDVybraneKarty ulož ID vybrané karty. 4. Zobraz detail karty dle p.IDVybraneKarty. Do p.CenaKarty ulož cenu karty. 5. Uživatel klikne na Storno 5.1. Opakovat krok 1 6. Uživatel klikne "Koupit" 6.1. Z tabulky Users vyber stav konta a ulož do p.Konto 6.2. Zkontroluj, zda p.Konto - p.CenaKarty > 0 6.3. Pokud není, zobraz zprávu o nedostatečném zůstatku. Pokračuj krokem 4. 7. Vlož nový záznam do tabulky CardsOwnership, kde user_id = ID přihlášeného uživatele a id_karty = p.IDVybraneKarty 8. Zobraz stav o průběhu operace.
Výstupní podmínky: žádné
Alternativní tok: žádný

Tabulka 2: Minispecifikace - koupě karty

Třídní diagram

Je zde zachyceno využití tříd Frameworku ve spolupráci s třídami vlastními. Dále je patrné dělění dle víceúrovňové architektury na kód patřící datové, aplikační a prezentační vrstvě.



Obrázek 16: Schéma vztahů mezi třídami jednotlivých vrstev

Uživatelské rozhraní

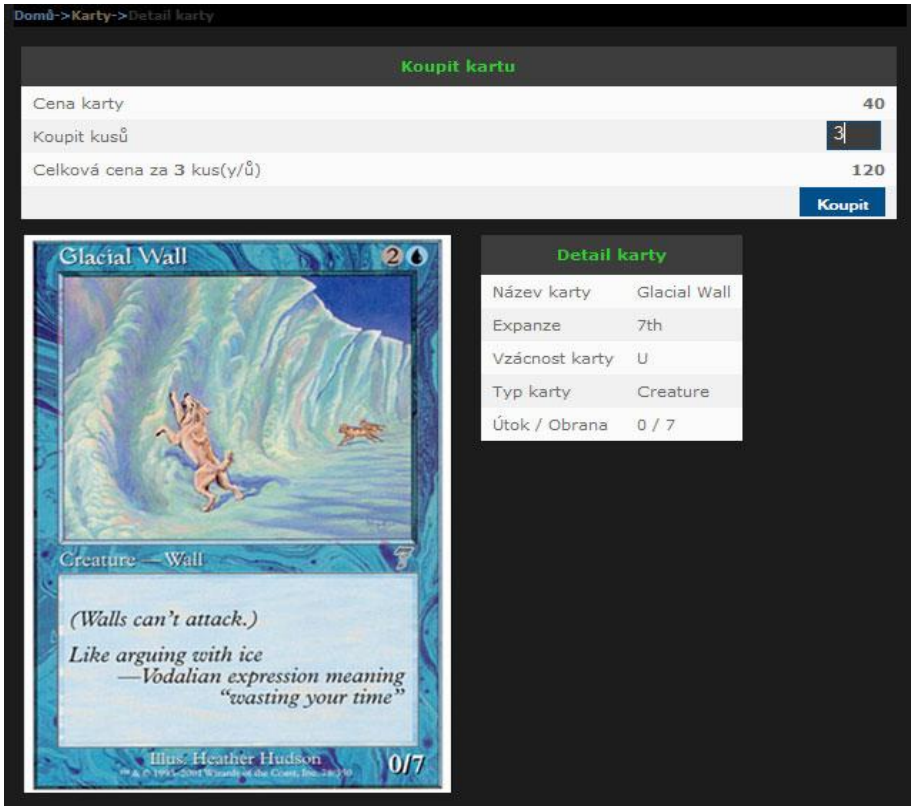
Uživatel si vyhledá kartu v seznamu a poté může přejít na její detailní popis. Pokud je uživatel přihlášen, je mu navíc nabídnuta možnost kartu si koupit (viz Obrázek 17: UI - Koupě karty). Při změně počtu kusů dojde k přepočítání celkové ceny. A konečně při stisku tlačítka koupit se zavolá metoda *btnBuy_Clicked()*. V metodě se nejprve zkontroluje, zda je správně zadán počet kusů, poté se zkontroluje, zda má uživatel dost finančních prostředků. Pokud je vše v pořádku, dojde k volání metody aplikační vrstvy *BuyCard()*. Na základě obdržené odpovědi dojde k informování uživatele.

```
protected void btnBuy_Clicked(object sender, EventArgs e) {
            Page.Validate();
            if (!Page.IsValid)
                return;

            int credit = profile.Games.VirtualMoney;
            int total = Int32.Parse(txtTotalCost.Text);
            int pieces = int.Parse(txtPieces.Text);

            if ((credit - total) < 0) {
                lblError.Visible = true;
                return;
            }

            lblSuccess.Visible = BasicCard.BuyCard(CardId, pieces);
                                        ...
}
```



Obrázek 17: UI - Koupě karty

Aplikační vrstva

Ve vrstvě aplikační logiky dojde k volání statické metody třídy *BasicCard*, která předá požadavek vrstvě datové. Jelikož datová vrstva neví nic o aktuálně přihlášeném uživateli, je předána i jednoznačná identifikace uživatele. Aplikační vrstva si počká na odpověď. Pokud převod karty proběhl úspěšně, jsou uživateli odečteny peníze a na závěr je o celém průběhu informována vrstva UI.

```
public static bool BuyCard(int cardId, int cardAmount)
{
      bool success = CardsProvider.BuyCard(cardId,
                                            cardAmount,
                                            CurrentUserID());
      var card = GetCardByID(cardId);

      if(success) {
      CurrenUserProfile.Games.VirtualMoney -= card.Price*cardAmount;
      }

      return success;
}
```

Datová vrstva

Zde teprve se provede samotná komunikace s databází. Nejprve se zkontroluje, zda již uživatel nevlastní kartu. Pokud ano, dojde pouze k navýšení tohoto počtu. Pokud ne, dojde k vytvoření nového záznamu. Na konec opět informuje vyšší vrstvu o stavu transakce.

```
internal static bool BuyCard(int cardId, int cardAmount, Guid userId)
{
      mtg_UsersHaveCard usersHaveCard = null;
      int amount = (from c in db.mtg_UsersHaveCards
                    where c.card_id == cardId && c.user_id.Equals(userId)
                    select c).Count();
      if (amount == 0) {
            usersHaveCard = new mtg_UsersHaveCard {
                  card_id = cardId,
                  amount = cardAmount,
                  user_id = userId
            };

            db.mtg_UsersHaveCards.InsertOnSubmit(usersHaveCard);
      }
      else {
            usersHaveCard = db.mtg_UsersHaveCards.First(uhc  =>
                  uhc.card_id == cardId &&
                  uhc.user_id.Equals(userId));

            usersHaveCard.amount = usersHaveCard.amount + cardAmount;
      }

      try { db.SubmitChanges(); }
      catch (Exception) { return false; }

      return true;
}
```

6.4. Využití Frameworku

Při využívání již hotových komponent, které pro vývojáře připravila třetí strana, se neobejdeme bez kvalitní dokumentace. Ani v mém případě tomu není jinak. Při používání serverových uživatelských prvků jsem aktivně pracoval s knihou (7), kde autor podrobně popisuje jednotlivé komponenty a uvádí jejich základní použití na jednoduchých, snadno pochopitelných příkladech.

- **Registrace uživatelů –** pro registraci uživatelů je použit ovládací prvek ASP .NET 2.0 s názvem *CreateUserWizard*. Vhodnou editací tohoto prvku dosáhneme vlastního požadovaného vzhledu registračního formuláře. Úpravou šablony můžeme uživateli zobrazit formuláře s údaji, které požadujeme zadat. Uživatel je nejprve vyzván k vyplnění povinných atributů jako uživatelské jméno, heslo a email. Po přesunu na další formulář je nejprve vytvořen účet a je vygenerován nový uživatelský profil, ve kterém jsou mimo jiné uchovávány nepovinné informace, které si může uživatel sám editovat.
  Komponenta je navržena tak, aby maximálně ochránila uživatelovy osobní údaje. Heslo není uloženo v čistě textovém formátu, ale je uloženo pomocí jednocestného hashovacího algoritmu SHA1. Přihlašovací údaje, uživatelské role a údaje profilu jsou rozděleny do více tabulek, tím se značně zkomplikuje možnost případného útoku.

- **Autentizace, autorizace uživatelů –** pro přihlašování uživatelů je použit ovládací prvek ASP .NET s názvem *Login*. Opět jsem použil vlastní šablonu pro definici vzhledu komponenty. Podobně jako v předchozím případně jsem použil i vlastní validační logiku pro ověření zadaných informací před odesláním požadavku na server. Samotné ověření zadaných uživatelských údajů provede běhové prostředí .NET pomocí metod standardních tříd, jako např. *Membership*.

- **Navigace –** prostředí ASP .NET nabízí některé řídící prvky, které umožní vytvořit navigační systém, který není napevno zapsán v HTML kódu stránek a je snadno udržovatelný. Nejprve je nutné definovat mapu webu do speciálního xml souboru s názvem *sitemap.xml*, kde uvedeme stromovou strukturu všech stránek aplikace. V konfiguračním souboru poté nastavíme, která mapa se má primárně používat. Menu se vytvoří pomocí dalšího řídícího prvku s názvem *Menu*, kterému opět přiřadím vlastní styl. Soubor s mapou webu se dá použít nejen pro tvorbu menu, ale také např. pro zobrazení navigační cesty, která informuje uživatele o tom, kde se zrovna nachází.

7. Implementace

Pro vývoj webové aplikace jsem používal integrované vývojové prostředí (IDE) Microsoft Visual Studio 2008 + SP1. Toto IDE nabízí podporu pro programovací jazyk C# 3.5, ale také JavaScript. Jazyk C# od verze 3.5 disponuje některými vlastnostmi, které usnadňují implementaci a zpřehledňují zápis. Z těchto vlastností jsem využíval především anonymní třídy, automatické property, lambda výrazy a integrovaný dotazovací jazyk LINQ. Při psaní kódu jsem ocenil zejména nástroj IntelliSense, který dokáže velmi inteligentně nabízet metody vhodné v daném kontextu, a tím snižuje množství překlepů a čas strávený při jejich hledání. Pro tvorbu „persistentního spojení" jsem využil doplněk Ajax Control Toolkit, který obsahuje několik uživatelských prvků aktivně pracujících s technologií AJAX. Visual Studio 2008 umí, jako doposud jediné IDE, ladit nejen kód psaný v některém z jazyků rodiny .NET, ale také klientské skripty psané v jazyce JavaScript.

Jako RDBMS je použit SQL Server 2008, který je podporován již zmíněnou technologií LINQ To SQL, která generuje samotný kód pro manipulaci s daty ve formě T-SQL. Veškeré atomické operace nad daty jsou transakčně bezpečné, takže nedochází ke ztrátám integrity v případě chyb. Výhodou je opět výborná integrace ve Visual Studiu.

Visual Studio obsahuje také vlastní webový server, který jsem ovšem nepoužíval. V knize (3) jsem se dočetl, že integrované běhové prostředí není dokonalé a od plné verze se liší například při využívání bezpečnostních politik. Proto jsem již při vývoji raději sáhl po ostré verzi IIS 7. Díky tomu jsem mohl vyzkoušet i přístup z internetu prostřednictvím své statické IP 89.29.108.22:80, kde aplikace prozatím stále běží. Po dokončení se bude registrovat nová doména, o jejímž názvu ještě prozatím není definitivně rozhodnuto.

Pro běh aplikace bude z počátku využit klasický webový hosting, protože není počítáno s přístupem mnoha uživatelů současně. V případě zvýšeného zájmu by bylo vhodné aplikaci nasadit na vlastní server.

8.Závěr

Jde o první projekt takového rozsahu, se kterým jsem se při svém dosavadním studiu setkal. Při vývoji této aplikace jsem si vyzkoušel kompletní řešení zadaného problému. Z tohoto důvodu nepředpokládám, že výsledné dílo nebude obsahovat žádné chyby. Mým cílem bylo navrhnout aplikaci tak, aby bylo možné jednoduše rozšířit jeho funkcionalitu o nové moduly.

Velký informační systém není možné vyvíjet metodou tzv. extrémního programování, tedy bez předchozí analýzy. Proto jsem se snažil využít metodik unifikovaného pracovního postupu (RUP). Při analýze celého systému jsem použil vývojové diagramy modelované pomocí jazyka UML, jako diagramy případů užití, třídní diagramy, nebo diagramy stavové.

Při samotné implementaci jsem se snažil co nejvíce využít již hotových komponent. Dle mého názoru není cílem psát veškerý kód znovu, ale mnohem lepší je použití zdokumentovaných prvků. K tomu mi nejvíce posloužil aplikační rámec ASP .NET 3.5. Dále jsem si vyzkoušel použití nových technologií, především nový jednotný dotazovací jazyk LINQ, který je k dispozici ve Frameworku .NET od verze 3.5.

Aplikace bude později rozšířena o nové karty se složitějšími vlastnostmi, které přibývají v nově vydávaných edicích. Dále by mohlo být vhodné implementovat prodej uživatelových karet formou aukce, která by umožnila lepší zhodnocení finančních prostředků v rámci webu.

Literatura

1. **ARLOW, Jim a NEUSTADT, Ila.** *UML a unifikovaný proces vývoje aplikací.* : Computer Press, 2003.

2. **BELLINASO, Marco.** *Webové programování v ASP.NET 2.0.* : Computer Press, 2008.

3. 101 LINQ Samples. *Ukázky použití LINQ.* [Online] 11 2008. http://msdn.microsoft.com/en-us/vcsharp/aa336746.aspx.

4. **Jehl, Scott a Hostetler, Mike.** Tutorials. *jQuery - Javascript API.* [Online] 2 2009. http://docs.jquery.com/Tutorials.

5. **STEPHENSON, Sam a Team, Prototype.** Prototype 1.5.1 The Complete API Reference. *Prototype Javascript API.* [Online] 1 2009. http://prototypejs.org/assets/2007/6/20/prototype-151-api.pdf.

6. **MACDONALD, Mathew.** *Beginning ASP .NET 3.5 in C# 2008.* : Apress, 2008.

7. **PROSISE, Jeff.** *Programování v Microsoft .NET.* : Computer Press, 2003.

8. Dokumentace. *ASP .NET AJAX.* [Online] 3 2009. http://www.asp.net/ajax/documentation/.

Přílohy

[I.]	datovy_slovnik.pdf	Kompletní datový slovník
[II.]	pravidla_mtg.pdf	Pravidla hry
[III.]	programátorská_příručka.pdf	Programátorská dokumentace